

From preprocessing and quality control to
biomarker discovery: challenges, risks and
opportunities of automated and data-driven
cytometry data analyses

Philippe Hauchamps
Promoter: Prof. Dr. Laurent Gatto
Co-promoter: Dr. Dan Lin

Computational Biology and Bioinformatics lab,
de Duve Institute,
UCLouvain,
1200 Brussels, Belgium

*It's never too late to be what you might
have been.*

—George Eliot

*Learning is not attained by chance, it
must be sought for with ardour and
attended to with diligence.*

—Abigail Adams

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my promotor, Laurent Gatto, for his unwavering dedication, daily support, and kindness throughout this PhD. His mentorship has enriched my work through countless discussions, practical tips, and insightful suggestions.

I am also deeply grateful to my co-promotor, Dan Lin, who made it possible for me to undertake this journey with GSK. Her continuous guidance, expert advice, and efforts to increase the visibility of my work through numerous presentations have been invaluable.

I also wish to thank the members of my steering committee - Pierre Coulie, Nicolas Dauguet, Bernadette Govaerts, Michel Verleysen, and Cyril de Bodt - for their guidance along the way. I am further thankful to my external jury members, Sofie Van Gassen and Samuel Granjeaud, for generously dedicating their time to assess this work.

Warm thanks go to GSK for funding my research over these four years, and to my colleagues at GSK. I am especially indebted to Stéphane Temmerman, who served as the initial senior sponsor of this project and provided invaluable support, advice, and recommendations throughout my time at GSK. I am equally grateful to Simon Delandre for providing essential datasets, as well as for the thought-provoking questions, challenges, and expert feedback during our many discussions. Special thanks go to Stéphanie Deroo - my publication manager at GSK - for her meticulous reviews and assistance with timely submissions. I also thank Walthere Dewe for his trust and senior-level support; Marie Toussaint, Babak Bayat, Mehdi Hamrouni, Sébastien Baudart, and Michaël Ska for facilitating access to key datasets and providing technical support; Sandra Nutti, Qing Yu, and George Elias for enriching discussions and knowledge sharing. Finally, I am grateful to my teammates Olajumoke Owokotomo, Oluwafemi Olusoji, and Mohammad Mehrian for their support,

collegiality, and for contributing to a positive and friendly working atmosphere.

I would also like to thank Samuel Granjeaud and Olivier Mendi-Coste from Inserm, who willingly acted as ‘expert guinea pigs’ for the *CytoMDS* package, providing encouraging and decisive feedback and suggestions.

I am grateful to the de Duve Institute for hosting me during this time, and to my labmates - Axelle Lorient, Chris Vanderaa, Julie Devis, Samuel Grégoire, Manon Martin, Lieven Desmet, Andrea Vicini, Enes Ayar, Riya Dogra, Guillaume Deflandre, Léopold Guyot, Margot Refuge, Shabnam Zaman, Matthieu Hodeige, and the e-Omix team: Jérôme Linden, Loïc Guille, and Anaghim Ftouhi - for their support, friendship, laughter, and the many good memories from conferences and team events. Special thanks to Andrea and Shabnam for the fruitful and collegial collaboration on the *MDSvis* Shiny application and the corresponding article. I also want to thank Badr-Ali Mouaden, my 2022–2023 Master student, for his early contributions and exploratory work that helped shape the direction of Chapter 10.

Thanks are also due to the European Bioconductor Society and the Association Française de Cytométrie (AFC) for awarding me travel grants, and to Prof. Marc Robinson, of Universität Zürich, for welcoming me into his team during a two-day hackathon in Switzerland.

Finally, this eight-year journey - including four years of a Master in Biostatistics - would not have been possible without the constant encouragement of my family and friends. I am especially indebted to my spouse, Alice, and my sons, Julien and Maxime, for their boundless patience, love, and unwavering support. Words cannot fully express my gratitude to them.

List of published works

This work has contributed to the following publications:

Peer-reviewed publications

- Hauchamps, Philippe, Babak Bayat, Simon Delandre, Mehdi Hamrouni, Marie Toussaint, Stéphane T. Temmerman, Dan Lin, and Laurent Gatto. 2024. “CytoPipeline and CytoPipelineGUI: A Bioconductor R Package Suite for Building and Visualizing Automated Pre-Processing Pipelines for Flow Cytometry Data.” *BMC Bioinformatics* 25 (1): 80. DOI: 10.1186/s12859-024-05691-z.
- Hauchamps, Philippe, Simon Delandre, Stéphane T. Temmerman, Dan Lin, and Laurent Gatto. 2025. “Visual Quality Control with CytoMDS, a Bioconductor Package for Low Dimensional Representation of Cytometry Sample Distances.” *Cytometry. Part A: The Journal of the International Society for Analytical Cytology* 107 (3): 177–86. DOI: 10.1002/cyto.a.24921.

Pre-print

- Hauchamps, Philippe, Andrea Vicini, and Laurent Gatto. 2025. “MDSvis, a Shiny Application for Interactive Visualisation of Multi Dimensional Scaling Projections.” *bioRxiv*. DOI: 10.1101/2025.06.23.660991.

Software packages

- CytoPipeline R package, released on Bioconductor project, version 3.17 - April 2023 <https://bioconductor.org/packages/release/bioc/html/CytoPipeline.html>
- CytoPipelineGUI R package, released on Bioconductor project, version 3.18 - October 2023 <https://bioconductor.org/packages/release/bioc/html/CytoPipelineGUI.html>

- CytoMDS R package, released on Bioconductor project, version 3.19 - April 2024 <https://www.bioconductor.org/packages/release/bioc/html/CytoMDS.html>
- MDSvis R package, released on github <https://github.com/UCLouvain-CBIO/MDSvis>

List of Abbreviations

1D	one-dimensional
2D	two-dimensional
acc	accuracy
ADC	Analog-to-Digital Converter
AI	Artificial Intelligence
AIDS	Acquired ImmunoDeficiency Syndrome
ANN	Artificial Neural Network
APC	AlloPhycoCyanin
APD	Avalanche PhotoDiode
AUC	Area Under the Curve
BAcc	Balanced Accuracy
C-index	Concordance Index
CCC	Concordance Correlation Coefficient
CMV	CytoMegaloVirus
CNN	Convolutional Neural Network
CPH	Cox Proportional Hazards
CyTOF	Cytometry by Time of Flight
DA	Differential Abundance
DE	Differential Expression

DL Deep Learning

DNA DeoxyriboNucleic Acid

DR Dimensionality Reduction

DS Differential State

EMD Earth Mover's Distance

F1 F1-score

FACS Fluorescence Activated Cell Sorting

FCS Flow Cytometry Standard

FITC Fluorescein IsoThioCyanate

FlowCAP Flow Cytometry: Critical Assessment of Population identification methods

FSC Forward SCatter

GPU Graphical Processing Unit

GUI Graphical User Interface

HIV Human Immunodeficiency Virus

HPC High Performance Computing

ImmPort Immunology database and analysis Portal

IPCW Inverse Probability of Censoring Weighted

LOO Leave-One-Out

MAD Median Absolute Deviation

MAE Mean Absolute Error

MDS Multi-Dimensional Scaling

MDSC Myeloid Derived Suppressor Cells

MFI Median Fluorescence Intensity

ML Machine Learning

MLP Multi Layer Perceptron
MSE Mean Squared Error
MST Minimal Spanning Tree
NGS Next Generation Sequencing
NIH National Institutes of Health
NPV Negative Predictive Value
OLS Ordinary Least Square
OMIP Optimised Multicolour Immunofluorescence Panel
PBMC Peripheral Blood Mononuclear Cell
PCA Principal Component Analysis
PCR Polymerase Chain Reaction
PE PhycoErythrin
PMT PhotoMultiplier Tube
PPV Positive Predictive Value
preci precision
QC Quality Control
RAM Random Access Memory
RBC Red Blood Cell
ReLU Rectified Linear Unit
RHS Right Hand Side
RMSE Root Mean Squared Error
RNA RiboNucleic Acid
ROC Receiver Operating Characteristic
rSD relative Standard Deviation
sensi sensitivity

SGD Stochastic Gradient Descent

SOM Self-Organising Map

speci specificity

SSC Side SCatter

TOF Time of Flight

tSNE t-distributed Stochastic Neighbour Embedding

UMAP Uniform Manifold Approximation and Projection

XAI eXplainable Artificial Intelligence

Preamble

In autumn 2019, I was starting the final year of my Master's degree in Biostatistics, nearing the end of a four-year journey to change careers after working for over 20 years in the banking sector. During my internship at the Computational Biology and Bioinformatics (CBIO) lab of the de Duve Institute, my supervisor, Prof. Gatto, invited me to take part in a collaboration with GSK. The objective of this collaboration was to explore automated methods for analysing flow cytometry data. At the time - after more than a decade of active research - the field had reached a certain level of maturity, and GSK was interested in assessing how modern computational tools could enhance their vaccine research and development efforts.

On my side, the project felt like a perfect opportunity: a chance to begin a PhD, to dive into immunology (which I knew almost nothing about at the time), to apply statistics and computer science methods to health-related research, and to contribute to open and reproducible science - something I care about deeply and which Prof. Gatto, a long-term contributor to the Bioconductor project, strongly embodies. After a period of planning and setup, I officially started my PhD in September 2021, and the present thesis is the result of that journey.

This work is situated within the field of computational cytometry, which lies at the intersection of immunology, high dimensional cytometry, and data science. Over the past two decades, flow and mass cytometry have revolutionised scientists' ability to characterise immune cell populations at single-cell resolution, generating complex datasets that require advanced computational approaches for effective analysis. As the volume and complexity of cytometry data have grown, so has the need for reproducible, scalable, and automated analysis methods that can support both exploratory research and clinical applications. My work contributes to this evolving field by developing and evaluating computational tools for the automated analysis of cytometry

data. Throughout this project, I have worked on methodological development, software implementation, and applied case studies, with a particular emphasis on the specific challenges and potential pitfalls associated with automated data analysis approaches.

This thesis is organised into three parts.

Part I provides contextual and foundational material to support the understanding of the results presented later.

- Chapter 1 outlines the principles of flow cytometry and its biomedical applications, describes the structure and content of the acquired data, and introduces two major technological advancements: mass cytometry and spectral flow cytometry.
- Chapter 2 presents an overview of key Machine Learning (ML) concepts. Since several automated data analysis methods rely heavily on ML techniques, this chapter provides essential background for the subsequent chapters.
- Chapter 3 offers a more detailed explanation of Deep Learning (DL), with a particular focus on Convolutional Neural Networks (CNNs), serving as a conceptual foundation for Chapter 10.
- Chapter 4 reviews the cytometry data analysis workflow, describing the main components of a typical analysis pipeline. Particular attention is given to preprocessing, Quality Control (QC), and low dimensional projection methods, as these are central to the contributions of this thesis.
- Chapter 5 introduces the software tools used for cytometry data analysis. It also includes a brief overview of the Bioconductor platform and sketches the principles of open science and reproducible research in computational biology, a recurrent theme throughout this thesis.
- Finally, Chapter 6 outlines the objectives and scope of the research presented in this thesis.

Part II details the research results and consists of four chapters.

- The first three chapters correspond to research articles that are published or under review in peer-reviewed journals. References to both the journal versions and the corresponding preprints - ensuring open access

- are provided. Each article chapter is preceded by a short introduction describing the motivation and context of the work.

- Chapter 7 presents the *CytoPipeline* suite, composed of two R packages designed to support the design, implementation, and evaluation of cytometry preprocessing pipelines.
- Chapter 8 introduces *CytoMDS*, a method for generating low dimensional projections of sample-level cytometry data using the Earth Mover's Distance (EMD), enabling visual identification of data quality issues across an entire study.
- Chapter 9 describes *MDSvis*, an interactive Shiny application built on top of *CytoMDS*, which allows users to explore Multi-Dimensional Scaling (MDS) projections in an intuitive and dynamic manner.
- Chapter 10 details the implementation of *CytoCNN*, a Python framework developed as part of this work, which applies deep CNN models to predict clinical outcomes from cytometry data. This chapter also discusses three use cases based on different datasets and outcome types.

Finally, **Part III**, made of a single chapter, concludes the thesis. In Chapter 11, we summarise the main scientific contributions of this work, highlight potential directions for future research, and provide some concluding reflections.

Contents

I	Introduction	1
1	Introduction to flow and mass cytometry	3
1.1	Principle and components of a flow cytometer	3
1.2	Sample data matrices	5
1.3	Sample preparation	7
1.4	Fluorescence, spectral overlap and compensation	8
1.5	Panel design	11
1.6	Biomedical applications	11
1.7	Technological evolutions	13
2	Introduction to Machine Learning	19
2.1	What is Machine Learning?	19
2.2	Model fitting and generalisation	20
2.3	Supervised vs. unsupervised learning	22
2.4	Types of outcome	22
2.5	Validating and testing the model	23
2.6	Model evaluation metrics	25
3	Introduction to Convolutional Neural Network models	35
3.1	Artificial Neural Networks	35
3.2	Network training	38
3.3	Convolutional Neural Networks	43
3.4	Dealing with overfitting	47
4	Cytometry data analysis	51
4.1	Overview of data analysis workflow	51
4.2	Positioning of cytometry data among single-cell omic data types	52
4.3	Preprocessing and quality control	54
4.4	Data visualisation using low dimensional projection	72
4.5	Cell population identification	83

4.6	Statistical analysis	90
5	Bioinformatics infrastructure and software	93
5.1	Cytometry data storage	93
5.2	Turnkey software solutions for cytometry data analysis	95
5.3	Open science and reproducible research with Bioconductor	96
6	Purpose of the research	99
II	Research projects	119
7	CytoPipeline & CytoPipelineGUI: a Bioconductor R package suite for building and visualising automated pre-processing pipelines for flow cytometry data	121
7.1	Context	121
7.2	Article	122
8	Visual quality control with CytoMDS, a Bioconductor package for low dimensional representation of cytometry sample distances	147
8.1	Context	147
8.2	Article	148
9	MDSvis, a Shiny application for interactive visualisation of Multi Dimensional Scaling projections	163
9.1	Context	163
9.2	Article	163
10	CytoCNN: a CNN framework for predicting clinical outcomes using cytometry data	171
10.1	Introduction and context	171
10.2	The <i>CytoCNN</i> model framework	174
10.3	Proof of concept: predicting CMV status using mass cytometry data	178
10.4	Predicting time to progression to AIDS for HIV+ patients	184
10.5	Predicting the age of individuals using flow cytometry data	198
10.6	Overall conclusions	214

III	Discussion	221
11	Summary, conclusion and perspectives	223
11.1	Summary of scientific contributions	223
11.2	Conclusion and perspectives	229
IV	Appendices	235
A	Concordance Correlation Coefficient (CCC)	A1
A.1	The concordance correlation coefficient for a bivariate distribution	A1
A.2	Formulas for n independent pairs of samples	A2
B	Supplementary results for Chapter 7	A5
C	Supplementary results for Chapter 8	A15
D	Supplementary results for Chapter 10	A27

Part I

Introduction

Chapter 1

Introduction to flow and mass cytometry

Flow cytometry is a high-throughput single-cell technology that enables simultaneous measurement of multiple physical and molecular characteristics of cells in suspension [1]. Originally developed in the 1960s [2] and refined through advances in optics, electronics, and fluorescent chemistry [3], it has become an indispensable tool in immunology, oncology, vaccine development, and clinical diagnostics.

This chapter starts with an overview of the principles of flow cytometry, focusing on the transformation of immune cell phenotypes into analysable digital data - Section 1.1. In Section 1.2, we describe the structure of the obtained data. In Section 1.3, we sketch some important steps of sample preparation. Sections 1.4 and 1.5 are devoted to considerations about fluorescence spectral overlap and panel design, respectively. Then, Section 1.6 outlines some of the main biomedical applications of flow cytometry. Finally, in Section 1.7, we address two major technological evolutions: spectral flow cytometry and mass cytometry.

1.1 Principle and components of a flow cytometer

The general principle of flow cytometry consists in labelling individual cells in a fluid suspension by fluorochrome-conjugated antibodies that bind to specific protein markers. As cells pass one by one through a laser beam, the fluorochromes are excited and emit light at specific wavelengths, which is detected and measured. This allows for the identification and quantification of distinct cell populations based on their marker expression. These markers are

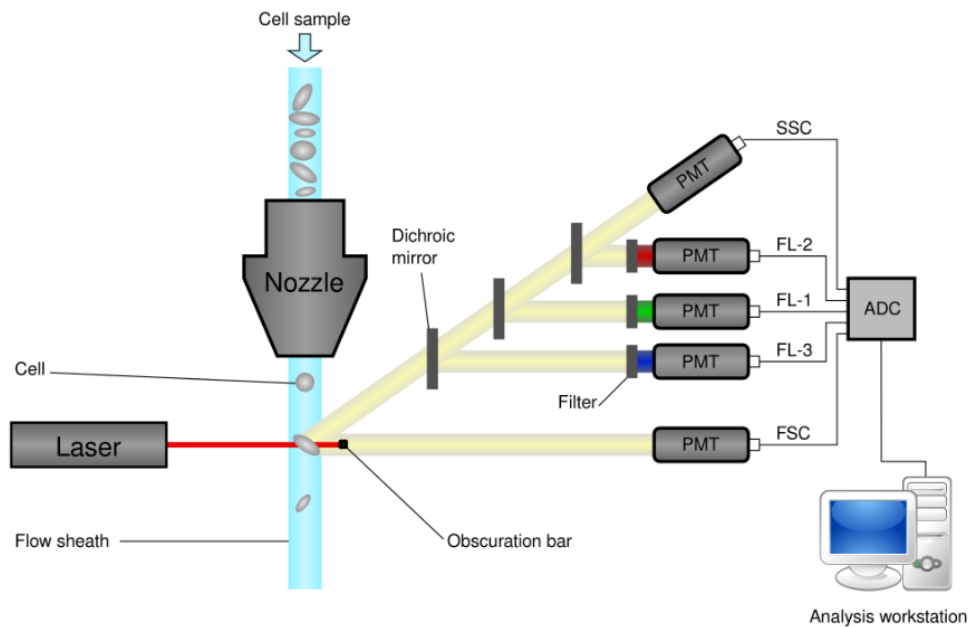


Figure 1.1: Schematic diagram of a conventional flow cytometer, showing the fluidic system allowing hydrodynamic focusing of the fluid sheath, the optical components - laser, dichroic mirrors and band-pass filters, the electronic components - Photo-Multiplier Tubes (PMTs) and Analog-to-Digital Converters (ADCs), and an analysis computer. For simplification, a single laser is represented, whereas modern instruments can have up to 5 lasers. Figure reproduced from [5].

mainly proteins expressed at the cell membrane surface. However, some intracellular markers, as well as cytokine secretion, can also be measured by flow cytometry, thanks to specific sample preparation protocols - Section 1.3.

A flow cytometer is made of three major components: fluidics, optics and electronics (Figure 1.1).

1.1.1 Fluidics

The fluidic system ensures that cells are presented to the laser beam one at a time. This is achieved by hydrodynamic focusing, where the sample is injected into a sheath fluid that constrains the cells into a single-file stream [4]. This configuration is essential to avoid cell doublets - as much as possible - and ensure accurate *event* detection. Events represent occurrences of particle

detection by the flow cytometer¹.

1.1.2 Optics

The optical system comprises excitation sources - usually solid-state lasers, as well as lenses, mirrors, and optical filters. As a particle passes through the laser beam:

1. it scatters light in different directions. Two main types of scattered light are measured: Forward SCatter (**FSC**), which is roughly proportional to cell size, and Side SCatter (**SSC**), which increases with cell content granularity, e.g. organelles or granules.
2. it emits fluorescence, if it has been labelled with antibodies conjugated to fluorochromes whose excitation spectra match the laser wavelength.

Light is collected, split according to its wavelength by a series of dichroic mirrors into detection pathways, and directed to photodetectors. In a conventional flow cytometer - as opposed to a spectral flow cytometer (see Section 1.7.2), each fluorochrome signal is detected in a separate channel based on its emission spectrum, thanks to channel-specific band-pass filters [6].

1.1.3 Electronics and signal processing

The photodetectors can be PhotoMultiplier Tubes (**PMTs**) or Avalanche PhotoDiodes (**APDs**). Analog signals from the detectors are amplified and digitised by Analog-to-Digital Converters (**ADCs**). More specifically, the **ADC** input signals have the shape of a pulse from which 3 features are extracted: its height, its width and its area. For example, for forward scatter, **FSC-H** denotes the pulse height, **FSC-W** the pulse width, and **FSC-A** the pulse area. Typically, for fluorescent light, only the area (-A) is kept for data analysis, while for scattered light channels (**FSC** and **SSC**), at least two parameters are kept. These multiple parameters are useful for cell doublet detection - see Section 4.3.5.

1.2 Sample data matrices

At the exit of the **ADCs**, the marker expressions of each event translate into a vector of digitised values corresponding to the intensity measured in the

¹In flow cytometry terminology, ‘event’ is often used as a synonym to ‘single cell’, but events can also correspond to debris, or cell doublets/multiplets passing through the laser beam.

	FSC-A	FSC-H	SSC-A	SSC-H	BV785 - CD3	APCy7 - CD4	BV650 - CD8a	APC - CD8b	AF700 - CD45RO	PerCP-Cy5.5 - CD45RA	L/D Aqua - Viability	BV605 - CD161
#1	119565	108322	82937	76142	4987.39	5014.57	264.22	-28.99	6499.16	1522.37	192.29	358.61
#2	107004	77596	262143	207144	1.63	2007.85	14.22	328.50	6773.95	370.09	690.67	452.46
#3	214908	139281	262143	226353	9344.98	7570.92	386.60	861.49	16512.68	2037.70	894.07	1365.85
#4	119453	107496	67842	62072	2570.97	14.94	14114.82	4053.07	-99.72	4819.84	204.99	287.68
#5	122345	111352	62728	56540	908.06	4161.10	203.41	-366.68	1377.33	-555.08	168.19	2389.44
#6	137030	124521	77133	70557	4192.33	174.56	22370.95	8093.75	2470.63	818.41	224.89	1352.73
#7	153801	122139	122174	80251	2490.87	157.03	376.82	-103.16	81.13	6020.49	597.93	1443.97
#8	177071	145150	168704	148426	122.76	750.26	154.81	-8.22	9477.76	529.41	656.41	679.26
#9	165853	118245	122113	89253	8206.78	10794.63	438.01	1150.37	15787.13	1055.24	359.88	7520.40
#10	121005	108937	72537	66157	13.12	32.66	-51.04	109.48	-175.57	3944.62	191.78	3085.20
#11	116611	106404	60610	56276	6623.25	5475.39	120.35	213.59	-1322.79	9521.15	150.82	715.57
#12	143652	114920	119107	105736	63.12	509.54	123.23	4.58	1809.89	976.23	285.11	180.31
#13	110192	101376	91706	83716	2015.14	317.63	9644.44	7377.56	512.94	3412.55	182.51	387.44
#14	102089	91005	77868	72361	2648.41	-122.42	6048.24	1945.53	470.43	1475.43	190.94	957.00
#15	130532	118882	75599	69650	2834.94	-107.14	9321.10	2161.09	1672.51	1430.40	206.57	502.29
#16	247445	114545	141314	74609	2725.71	127.31	9560.18	2085.40	2021.79	6238.87	421.97	1844.79
#17	134274	119368	84765	78348	125.54	84.80	1046.80	-242.33	574.93	7022.91	175.16	700.41
#18	101729	91521	53380	49425	4315.97	3923.75	275.34	-224.97	153.87	11911.57	118.30	567.69
#19	55896	54100	61738	56176	14.54	63.69	-275.12	159.44	68.69	7921.60	3931.65	2143.51
#20	115054	105932	82371	73452	3294.60	4718.42	154.41	473.51	2861.89	2883.44	183.10	438.80

Figure 1.2: Example of cytometry sample data matrices. Each sample corresponds to a single matrix, where events (cells) are organised in rows, and markers are organised in columns. For scattered light channels, 2 parameters of the pulse-shaped signal are retained - here the area (FSC-A, SSC-A), and the height (FSC-H, SSC-H). For fluorescent channels, only one parameter is retained - here the area, not mentioned in the column names. Here, fluorescent channel names are made up by concatenation of fluorochrome and marker names. In high dimensional flow cytometry, a typical sample matrix contains a few dozens of columns (markers) and millions of rows (events).

scatter (FSC/SSC), and fluorescence channels. These values are then stored in a digital matrix, as shown in Figure 1.2. Upon multiple sample data acquisitions, each cytometry sample corresponds to one data matrix, with marker parameters in columns and events - hopefully cells - in rows. In high dimensional flow cytometry, a typical sample matrix contains a few dozens of columns (markers) and millions of rows (events). These sample matrices are the raw data input to the data analysis process - see Chapter 4.

Protein markers measured in fluorescent channels are usually - but not always - denominated with names like ‘CDX’, where ‘X’ is a number between 1 and several hundreds. ‘CD’ stands for *Cluster of Differentiation*, a term that originates from the early 1980s. At the time, immunologists were in the process of standardising the naming of leukocyte surface molecules recognised by monoclonal antibodies. Therefore, antibodies with the same target were grouped into ‘clusters’, giving rise to CD numbers [7].

Markers can be classified into 3 categories:

- *Lineage* markers (e.g. CD3, CD4, CD8, CD14, CD19, CD56,...), which are used to define the belonging of a cell to specific immune cell popu-

lations (CD4 or CD8 T cells, monocytes, B cells, NK cells, ...). These are often considered as ‘on-off’ markers, of which the intensities show a bi-modal distribution.

- *State* markers (e.g. activation markers (e.g. CD69), memory markers (e.g. CD45RO), etc.), which characterise the state of specific cells during their life. Their expression intensity is already more ‘continuous’ - as opposed to ‘on-off’, showing a distribution that is not necessarily bimodal and/or gaussian.
- *Functional* markers (e.g. cytokine production (INF- γ , IL-2, TNF- α ,...), cell proliferation markers (Ki-67), ...), which are ‘quantitative’ markers, typically having a continuous domain of expression intensities. Moreover, their expression intensity values can sometimes be low (‘dim markers’), and/or not easily predictable, depending on the experimental context.

1.3 Sample preparation

Sample preparation is a critical step for producing high quality data. The goal is to obtain a single-cell suspension in which cells are intact, viable (if necessary), and appropriately labelled with fluorochrome-conjugated antibodies that bind to chosen surface or intracellular markers. Monoclonal antibodies are selected to target these markers. Surface markers (e.g. CD3, CD4, CD8) are stained live, while intracellular targets (e.g. FoxP3, cytokines) require fixation and permeabilisation - see below. Depending on the sample type - peripheral blood, bone marrow, or tissue - the preparation steps can vary significantly in complexity [8, 9].

1.3.1 Peripheral blood and bone marrow samples

Peripheral blood is one of the most straightforward sample type for flow cytometry. Blood is typically collected in anticoagulant-containing tubes to prevent clotting. Red Blood Cells (RBCs) must be removed or lysed to allow for efficient analysis of leukocytes. After lysis, the remaining white blood cells are washed and resuspended in an appropriate buffer to block nonspecific binding of antibodies to cell membranes. Alternatively, if granulocytes are not required, density gradient centrifugation (e.g. using Ficoll) can be used to isolate Peripheral Blood Mononuclear Cells (PBMCs), i.e. lymphocytes and monocytes. Staining is then performed by incubating the cells with fluorochrome-conjugated antibodies targeting surface markers.

Bone marrow samples are handled in the same fashion as peripheral blood samples. However, bone marrow aspirates are more complex than blood samples due to higher cellularity and the presence of immature progenitor cells. Because hematopoietic stem and progenitor cells are a focus in many bone marrow analyses, care must be taken to avoid cell loss or damage during lysis.

1.3.2 Tissue samples

Tissue samples (e.g. spleen, lymph nodes, liver, tumors, ...) require mechanical dissociation and/or enzymatic digestion to obtain a single-cell suspension [10]. After digestion, samples are filtered through a fine mesh (e.g. 40–70 μm) to remove debris and clumps. Depending on the tissue type, additional density gradient centrifugation (e.g. using Ficoll) may be needed. In the case of single-cell suspensions obtained from tissues, debris and dead cell removal steps during data preprocessing (Section 4.3.5) will be even more crucial.

1.3.3 Staining for intracellular markers

To detect intracellular proteins such as transcription factors (e.g. FoxP3), cells must be fixed and permeabilised after surface staining. Fixation - commonly with formaldehyde - preserves cellular structures, while permeabilisation - using detergents like saponin - allows antibodies to enter the cell and bind intracellular targets [11].

1.3.4 Measurement of cytokine production

Cytokine detection by flow cytometry requires capturing intracellular cytokines, as they are typically secreted after synthesis. This is accomplished through intracellular cytokine staining, which involves stimulating cells in the presence of protein transport inhibitors. These inhibitors trap cytokines inside the cell by blocking secretion. After stimulation (usually 4–6 hours), cells are fixed and permeabilised, and fluorochrome-conjugated antibodies are used to detect cytokines [11].

1.4 Fluorescence, spectral overlap and compensation

Fluorochromes are molecules that absorb light at specific wavelengths and re-emit at longer wavelengths. Examples include Fluorescein IsoThioCyanate

(FITC), PhycoErythrin (PE), AlloPhycoCyanin (APC), and tandem dyes² like PE-Cy7 (PhycoErythrin-Cyanine 7).

Unfortunately, the emission spectrum of a fluorochrome is not concentrated in a single bandwidth, but typically span a broader range of wavelengths. In the case of multicolour panels, this causes emission spectra overlap, as is shown in Figure 1.3 (a). This spectra overlap, also called *spill-over* or *cross-talk*, requires *compensation*, i.e. subtracting spill-over from the measured channel signal, in order to isolate the signal specific to each marker [12]. Compensation matrices are obtained using single-stained controls; i.e. samples that contain either compensation beads, a subset of cells from the samples under study, or a combination of both. These samples are stained with only one fluorochrome-conjugated antibody at a time, allowing to assess and record the amount of spill-over of the fluorochrome at hand, into all other channels. In modern flow cytometers, compensation is implemented as a piece of software, which constitutes a dedicated preprocessing step of the data analysis - see Section 4.3.1³.

On Figure 1.3 (b), one can notice the so-called *spreading error*, highlighted on the plots for the blue cell population - before and after compensation - with the corresponding relative Standard Deviation (rSD), a.k.a. coefficient of variation⁴. Spreading error is an intrinsic characteristic of flow cytometry measurements. It is a consequence of the imprecise measurement of fluorescent signals at the detector (typically a PMT), which show some variance due to uncertainty in photon counting [11]. While subtracting the spill-over, compensation makes this spreading error more apparent by bringing it to the lower scales. The consequence is a signal dispersion in the low and negative intensities, resulting in signal resolution loss.

²Tandem dyes are pairs of covalently linked fluorophores - a donor and an acceptor - that work together to broaden the spectral range detected using a single laser excitation, allowing for more markers to be analysed in a multi-colour panel.

³In earlier times of analog flow cytometers, compensation was performed by the electronic signal processing components, in real time during data acquisition.

⁴Actually, the range indicated on the plot is a multiple of the rSD.

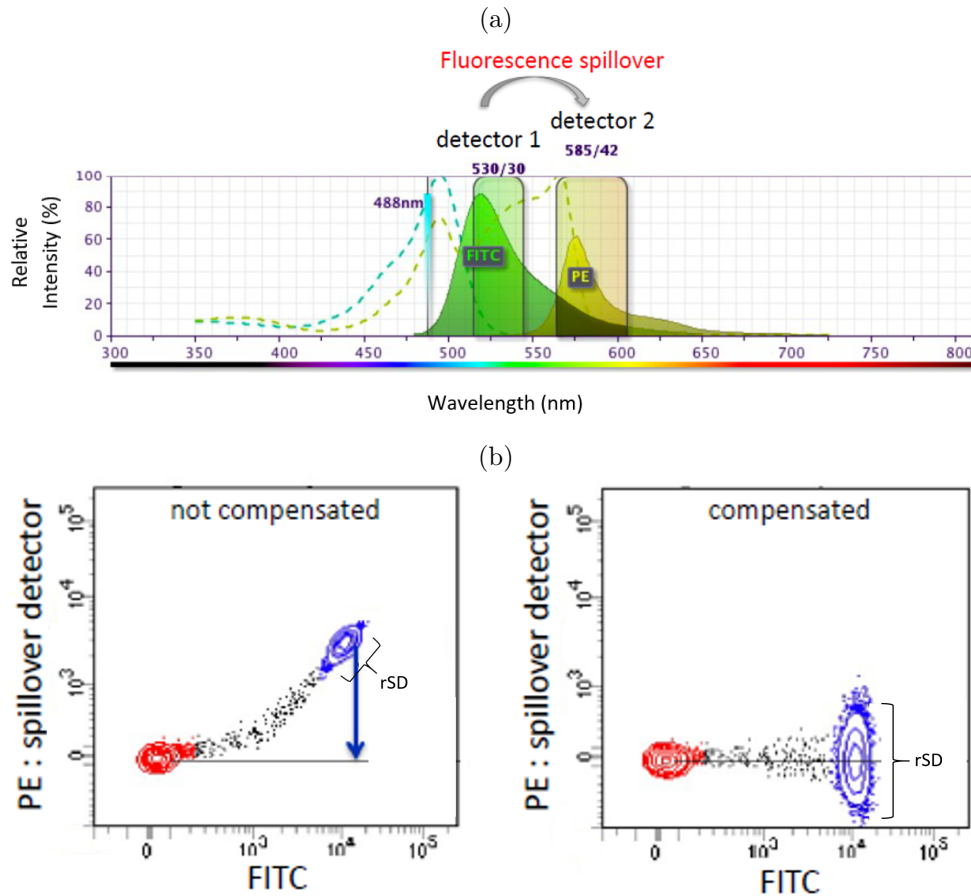


Figure 1.3: (a) The excitation spectra (dotted lines) and emission spectra (plain histograms) of two fluorochromes, i.e. **FITC** and **PE**. The selected laser wavelength - 488 nm - corresponds to both excitation spectra, and the **FITC** emission spectrum overlaps with the **PE** emission spectrum. Therefore, the signal observed in the **PE** channel - in the detector 2, centered on 585 nm - includes emission by the **FITC** fluorochrome. (b) Two-dimensional representation of events according to channel intensities observed for **FITC** (x axis) and **PE** (y axis). The red contours correspond to a double negative population, while the blue contours correspond to a single positive population, where only the marker corresponding to **FITC** is present. The left plot, prior to compensation, shows that the blue population is wrongly displayed as positive for **PE**. The right plot shows the events after compensation correction: the blue population is now only positive for **FITC**, and negative for **PE**. Figure adapted from N. Daguët, “flow cytometry basics training”, de Duve Institute/UCLouvain.

1.5 Panel design

A flow cytometry *panel* can be defined as a carefully designed combination of fluorescently labelled antibodies, each targeting a specific cellular marker, assembled to identify and characterise multiple distinct cell populations simultaneously during a single acquisition [13]. The art of *panel design* involves strategically selecting and assigning antibody-fluorochrome combinations to markers of interest, given the scientific question at hand. This should be done such as to minimise spectral overlap and optimise instrument sensitivity, given the instrument characteristics (notably the available laser lines).

High-parameter panels, i.e. with more than 20 colours, thus require extensive optimisation, including titration of antibodies⁵, iterative testing (pilot experiments), and well-designed controls. The design of these panels is therefore a labour-intensive task, which can take several weeks, or even months. Tools such as Optimised Multicolour Immunofluorescence Panels (OMIPs), published in Cytometry Part A journal [14], provide validated starting points and gating strategies for specific use cases, helping reduce design time and enhancing reproducibility.

1.6 Biomedical applications

In this section, we provide a non exhaustive list of flow cytometry biomedical applications.

1.6.1 Cell sorting

The most widespread use of flow cytometry is probably cell sorting. Indeed, flow cytometry can be used to physically isolate specific cell populations in a process called Fluorescence Activated Cell Sorting (FACS) [4]. During sorting, cells are analysed individually for fluorescence and scatter properties. Those meeting predefined criteria are electrically charged and deflected into separate collection tubes (Figure 1.4). This allows researchers to purify live cells with distinct phenotypes (e.g. CD4+ T cells) for downstream experiments such as cell culture, RiboNucleic Acid (RNA) sequencing, or functional assays.

⁵Titration of an antibody is the process of experimentally determining the optimal antibody concentration that provides the highest specific signal with the lowest background, ensuring accurate and reproducible staining.

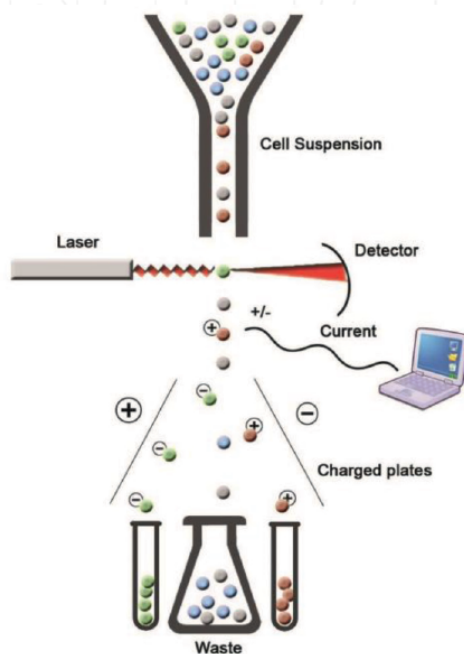


Figure 1.4: Principle of FACS. Figure reproduced from [15].

1.6.2 Applications in immunology

Flow cytometry has been the technology of choice for studying the immune system for the last four decades. Major applications in immunology include:

- immune cell phenotyping in health and disease
- measuring immune responses to pathogens
- studying host-pathogen interactions

1.6.3 Applications in cancer biology

Flow cytometry was the technology of choice for applications related to hematology cancers in the 1980s and 1990s. Nowadays, it is progressively replaced by other technologies, e.g. Polymerase Chain Reaction (PCR) or Next Generation Sequencing (NGS). Major applications include:

- studying cancer cells (phenotype, DeoxyriboNucleic Acid (DNA) content)
- clinical diagnostic tests

- help in patient prognosis

1.6.4 Applications in vaccine development

Flow cytometry is ubiquitous in vaccine development. It is used at different development stages, from candidate selection, pre-clinical, as well as all phases of clinical trials. Its numerous applications include:

- in vitro potency assay to down select vaccine candidates (antigens and delivery systems)
- in vivo immunogenicity of vaccine candidates in animal models
- antigen-specific CD4/CD8 T cell responses
- memory T and B cell responses to vaccine formulations
- innate cells activation (reactogenicity)
- antibody functionality
- mechanism of action of vaccines
- vaccine dose response assessment
- systems vaccinology, i.e. the study of complex interactions within the immune system
- ...

1.7 Technological evolutions

In this section, we introduce two recent technological evolutions that have a significant impact on data analysis, which is the main subject of this thesis. The first one is spectral flow cytometry, which can be considered as a technological advance of multicolour flow cytometry. The second one is mass cytometry, an alternative high-throughput cytometry technology. Both aim at increasing the number of parameters (markers) that can be measured simultaneously for each cell.

1.7.1 Limitation of the number of markers in conventional flow cytometry

In what follows, we define as *conventional flow cytometry* the instrument technology that detects photons of different wavelength (colour) with individual photodetectors associated with specific optical filters [16].

The number of parameters (colours) that can be simultaneously measured in conventional flow cytometry is primarily determined by the number of photodetectors and their associated band-pass filters. A key constraint in expanding the number of detectable colours lies in the trade-off between using narrow band-pass filters - which reduce spectral overlap - and the corresponding reduction in the number of photons reaching the detector, which can compromise sensitivity. Typically, band-pass filters have widths of 20 to 50 nm, and since the total usable fluorescent emission spectrum ranges from approximately 350 to 850 nm (~ 500 nm), this spectral window can accommodate only about 10 to 12 distinct channels for detection in the UV, violet, and blue laser ranges, and even fewer for longer wavelength lasers such as yellow-green and red [16]. Modern flow cytometers often incorporate five to seven spatially separated lasers, each with adjustable power settings, theoretically enabling the discrimination of up to 40–50 parameters. However, in practice, the extent of spectral overlap between fluorochromes imposes significant challenges for accurate signal separation. As a result, the practical limit for reliable, multiparametric analysis typically remains around 30 simultaneous colours [17].

1.7.2 Spectral flow cytometry

The first prototype of a spectral flow cytometer was introduced in 2004 [18]. Unlike conventional instruments that rely on individual detectors and band-pass filters to isolate the dominant emission band of each fluorochrome, spectral flow cytometers capture the entire fluorescence emission spectrum of all dyes using an array of sensors. This method employs more detectors than fluorochromes, allowing each dye to be represented across multiple channels [17] - see Figure 1.5.

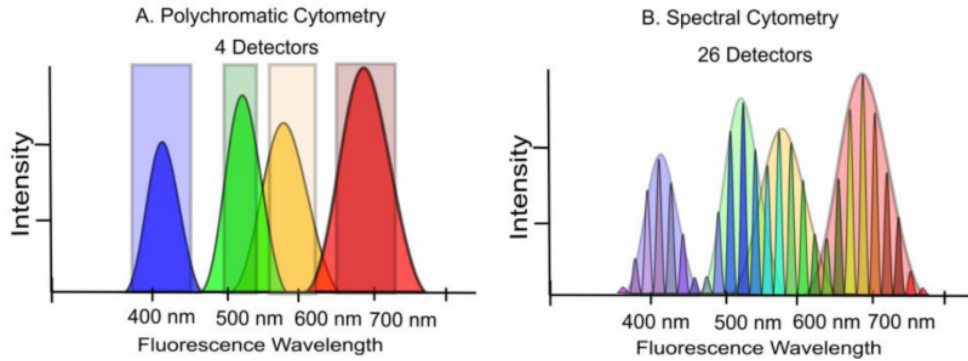


Figure 1.5: Comparison between conventional flow cytometry (panel A) and spectral flow cytometry (panel B). In conventional flow cytometry, each fluorescent dye emits light that is captured by a dedicated detector tuned to a specific wavelength range. In contrast, spectral flow cytometry uses multiple detectors to capture the full emission spectrum of all fluorochromes, allowing for a computational process known as ‘spectral unmixing’ to accurately distinguish each dye. Figure reproduced from [17].

Data analysis of samples acquired with spectral flow cytometry requires similar steps as the analysis of conventional flow cytometry samples. The main difference lies in the compensation method - referred to as (*spectral*) *unmixing* in the context of spectral flow cytometry. Unmixing takes advantage of the redundancy present in the spectral data, to allow compensation between signal of fluorescent dyes having relatively similar emission spectrum, as well as identification and separation of cellular autofluorescence signal - see Section 4.3.1.

1.7.3 Mass cytometry

Mass cytometry, also referred to as Cytometry by Time of Flight (**CyTOF**), was first introduced in 2009 [19]. Unlike conventional fluorescent flow cytometry, mass cytometry uses stable heavy metal isotopes instead of fluorescent dyes to label antibodies. This approach avoids, to a large extent, issues related to overlapping fluorescence spectra and significantly expands the number of parameters that can be measured simultaneously.

In mass cytometry, cells are stained with a panel of antibodies, each tagged with a distinct non-radioactive metal isotope (Figure 1.6). The labelled single-cell suspension is then nebulised, so that each droplet contains one cell, and introduced into an argon plasma. The intense heat atomises and ionises each cell, generating a cloud of ions that reflects the elemental composition of the cell surface or interior. A quadrupole filter eliminates low-mass biological ions,

leaving behind only the heavier metal isotope ions. These ions then enter a Time of Flight (TOF) mass analyser, where they are separated based on their mass-to-charge ratios. As they strike the detector, the ions generate electrical signals proportional to their abundance.

While, in theory, up to 120 parameters can be measured in parallel, practical constraints such as the availability of pure isotopes and reliable conjugation chemistries typically limit panels to around 60 parameters [20]. Apart from avoiding fluorescence spectra overlap, an additional benefit of mass cytometry is the near-complete absence of background signal due to cellular autofluorescence, which can be a significant source of noise in fluorescence-based flow cytometry.

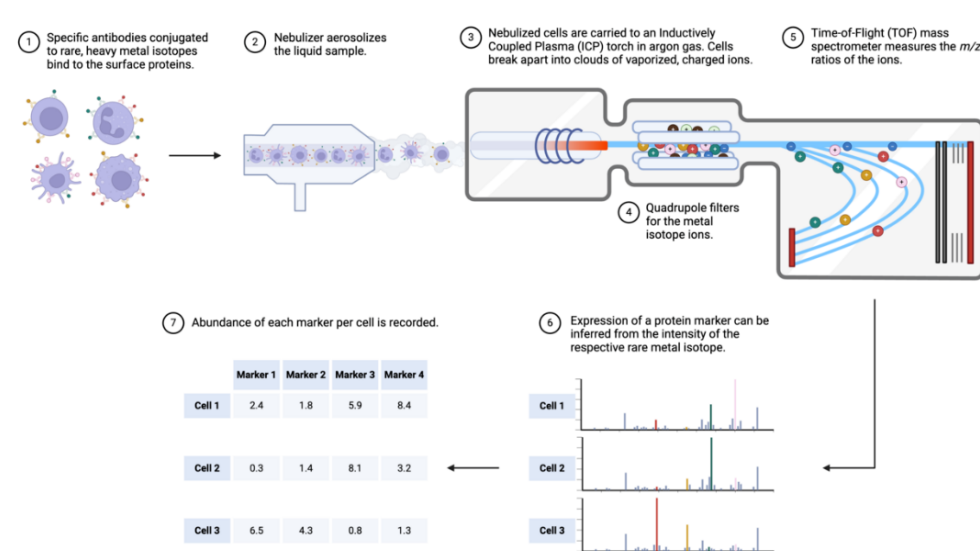


Figure 1.6: Workflow of mass cytometry (CyTOF) - figure adapted from <https://en.wikipedia.org/wiki/CyTOF>.

Despite these strengths, mass cytometry also presents some drawbacks. First, it is a destructive technique: cells are atomised in an argon plasma and cannot be recovered for downstream applications such as cell sorting or live-cell functional assays. This contrasts with flow cytometry, which permits live-cell sorting based on user-defined gates. Second, mass cytometry operates at a significantly lower acquisition speed, typically in the range of 100–1,000 cells per second, compared to 1,000–10,000 cells per second in flow cytometry. This limits its suitability for high-throughput screening or detection of rare

cell populations.

Three features of mass cytometry, as compared to conventional flow cytometry, have an impact on data analysis. First, as already mentioned, cross-talk between channels, while not completely absent [21], is strongly reduced, compared to flow cytometry. In most cases, this avoids applying any compensation to CyTOF data. However, in experiments where very high sensitivity is needed, a dedicated compensation method could still be used (see Section 4.3.1), to further reduce the residual cross-talk. Second, due to the absence of scattered light channels in mass cytometry, some of the preprocessing steps, e.g. debris, doublets and dead cells elimination, which are typically performed, in flow cytometry, by gating events in the FSC and/or SSC dimensions, need to be slightly revisited (see Section 4.3.5). And third, due to its slower acquisition time, combined with its large number of available channels, mass cytometry is particularly advantageous for multiplexing approaches, such as mass-tag cellular barcoding [22]. Multiplexing allows the simultaneous acquisition of multiple samples, which greatly contributes to decreasing sample-to-sample technical variation. Hence, the analysis of multiplexed mass cytometry data requires an additional *de-barcoding* preprocessing step - see Section 4.3.2.

Chapter 2

Introduction to Machine Learning

In this chapter, we review some concepts of Machine Learning (ML) that will prove useful for the understanding of subsequent chapters, notably Chapters 3, 4 and 10.

2.1 What is Machine Learning?

ML is a field of Artificial Intelligence (AI) that focuses on developing algorithms capable of learning from data and making predictions or decisions without being explicitly programmed [23]. In other words, ML models uncover patterns in data, allowing them to generalise from past observations (*training data*) to make informed decisions about new, unseen data. For example, a ML algorithm could be trained on historical weather data to predict future temperatures or on labeled medical images to identify the presence or absence of a disease.

Note that while ML is an important subset of AI, not all AI applications rely on ML. For instance, *expert systems*, another sub-class of AI applications, use rule-based programming to simulate the decision-making ability of a human expert [24]. In these systems, the rules that lead to the system decisions are explicitly programmed ‘beforehand’ by the system designers, and are therefore not ‘learned’ from the data, or through experiments performed by the system itself.

2.2 Model fitting and generalisation

Model fitting or *training* refers to the process of adjusting a model’s parameters so that it best captures the patterns in the training data. A model that fits the training data well will have a low training error. This error summarises observed discrepancies between model predictions and the corresponding true values for each training data point. However, good performance on the training data does not necessarily mean that the model will perform well on new, unseen data. *Generalisation* refers to a model’s ability to apply learned patterns from the training data to new data. A model that generalises well will have low error not only on the training set but also on a *test set*, i.e. some data not seen during training.

On the one hand, *underfitting* occurs when a model is too simple to capture the underlying structure of the data, leading to poor performance on both the training and test sets. For example, fitting a linear model to a clearly non-linear relationship would result in underfitting. On the other hand, *overfitting* occurs when a model is too complex, capturing noise or irrelevant patterns in the training data. Overfitting leads to low training error but poor performance on a test set because the model is too tailored to the training data. Addressing underfitting and overfitting requires finding a balance between model complexity and generalisation. This is also known as the bias-variance trade-off [25]. In this context, *bias* is to be understood as an error that is due to simplifying assumptions made by the model, while *variance* refers to the high sensitivity to small fluctuations in the training set. High bias leads to underfitting while high variance leads to overfitting.

Figure 2.1, reproducing an example taken from [26], illustrates these concepts. When choosing the degree of a polynomial regression to fit a one-dimensional (1D) dataset, a low-degree polynomial might underfit the data, while a high-degree polynomial might overfit. A moderate-degree polynomial would balance bias and variance, improving generalisation. Note that techniques such as regularisation, which penalises overly complex models, can come into play to mitigate the risk of overfitting (see Section 3.4.2).

To conclude this section, it is worth noting that recent studies have shown that the relationship between model complexity and generalisation can deviate from the classical bias–variance trade-off described above. In modern models containing a very large number of parameters, performance does not necessarily deteriorate beyond the overfitting regime but can improve again,

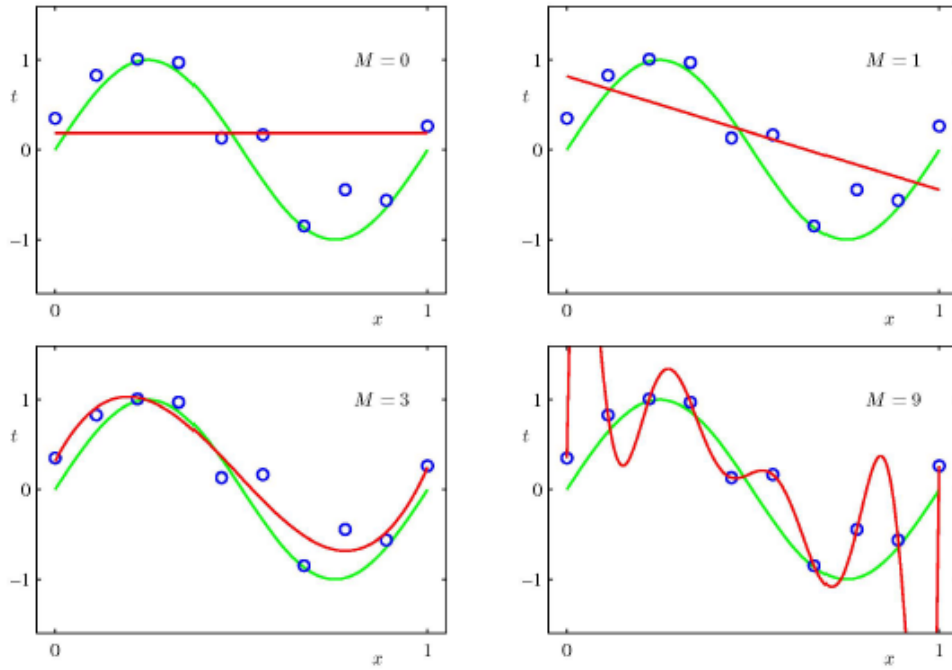


Figure 2.1: Regression with M -degree polynomial : different polynomial models (in red), corresponding to increasing degree M of polynomial, are fitted on the same training set (blue circles). The target function, used to generate the training data, is displayed as a light green curve. A too small degree of polynomial ($M = 0$ or $M = 1$) poorly fits the training set, leading to underfitting. At the other extreme, a polynomial of degree $M = 9$ perfectly fits the training set, but generalises poorly to unseen data points. A better trade-off is found but with a polynomial of degree $M = 3$, which does not fit the training set perfectly but improves generalisation. Figure reproduced from [26].

a behaviour referred to as *double descent* [27]. This phenomenon introduces a second regime beyond the interpolation threshold - defined as the point at which the model has enough parameters to fit the training data perfectly, where further increases in complexity can lead to improved generalisation.

2.3 Supervised vs. unsupervised learning

ML tasks are broadly divided into two main categories: *supervised* learning and *unsupervised* learning. In supervised learning, the model is trained on *labeled* data, meaning that the input data comes with corresponding target values (labels). The goal of supervised learning is to learn a mapping from inputs to outputs so that the model can predict the label of new, unseen data. For instance, a supervised learning task in the clinical domain could be the prediction of patient prognosis (the output), based on patient specific data like symptoms, patient history, results to specific tests, etc. (the inputs). In that case, the predicting model would need to learn the target relationship between inputs and outputs from a database of patients for which the outcome is already known, i.e. a labeled training set. In unsupervised learning, the model is trained on data without labels. The goal is to identify patterns, structures, or groupings in the data without predefined outcomes. The most common unsupervised learning tasks are *clustering*, i.e. grouping data points based on similarities, and Dimensionality Reduction (**DR**), i.e. reducing the number of features in a dataset while retaining essential information. In the remainder of this chapter, we will focus on supervised learning.

2.4 Types of outcome

Supervised **ML** tasks are typically split into the following types of problems:

Classification

In classification problems, the outcome variable is categorical (discrete), and the goal is to assign each data point to one of a finite set of classes. In *binary* classification, there are two possible classes (e.g. identifying malignant vs. non malignant tumors from tumoral tissue images). In *multiclass* classification, there are more than two possible classes (e.g. classifying patients among several possible diseases based on clinical symptoms and lab results).

Regression

In (standard) regression problems, the outcome variable is continuous, and

the goal is to predict a numeric value based on input features. One possible example is predicting drug concentration over time in pharmacokinetics, based on features like dose, body weight, time since administration, etc.

Survival prediction

In survival prediction, the outcome variable is the time until an event occurs (e.g. predicting survival time of patients suffering from a specific disease, or predicting time to relapse for cancer patients undergoing a specific treatment). Unlike in traditional regression problems, survival data include censored observations, i.e. cases where the event has not (yet) occurred during the observation time span. This data characteristic typically leads to specific mathematical treatments and models, as will be discussed in Sections 2.6.3 and 3.2.1.3.

2.5 Validating and testing the model

Assessing the generalisation capabilities of a trained model requires the use of data points that are not part of the training set. In ML terminology, these additional data points are called a *validation* dataset. In its most straightforward form, the validation set can be created by ‘setting aside’ a given portion (for example, 20%) of the available training data, and use only the remaining part for effective model training. A validation error can then be calculated based on a chosen metric, depending on the type of outcome (see Section 2.6).

In the ML context, one should also distinguish between model *parameters*, and model *hyperparameters*. Parameters are quantities of the models that are adjusted to the training dataset during the fitting (or learning) process. Hyperparameters are model quantities and characteristics that need to be chosen beforehand by the user, and are kept as is during training. In the polynomial regression example (Section 2.2), parameters are the coefficients of the regression polynomial (e.g. the slope and intercept in the case of a linear regression), while a hyperparameter is the chosen degree of the polynomial, M . In more complex models like Artificial Neural Networks (ANNs), hyperparameters can be numerous, e.g. the number of layers of the network, the number of nodes per layer, the activation function shape, the regularisation parameters, the learning rate,... (see Chapter 3).

Although not explicitly trained, these hyperparameters can have a major impact on the accuracy of the obtained model, hence they need to be carefully chosen. One possible avenue is to automatically tune them thanks to

some optimisation algorithms. For instance, a simple algorithm for finding good hyperparameter values is *grid search*, which can be done when the number of hyperparameters is small. It consists in selecting a manually specified subset of possible combinations of hyperparameter values, try out all combinations and finally select the best combination, giving the best model evaluation. However, when the number of hyperparameters grows, this technique becomes quickly too computationally expensive. In such situations, it may be necessary to resort to more advanced techniques, such as Bayesian optimisation [28]. This hyperparameter tuning, as well as possibly *model selection*, which consists in selecting the best model type between competing ML model approaches, are also performed as parts of model validation, by comparing the validation metric(s) obtained for the different model instances.

In case the initial training set is too small to afford putting aside a substantial portion of the data for model validation, dedicated approaches can be undertaken, e.g. bootstrap or cross-validation [29], the latter being the most popular approach. In its generic k -fold version, cross-validation consists in splitting the dataset in k subsets. The model is then trained on the union of $k - 1$ subsets and validated on the remaining one subset. This train/validation loop is repeated k times, with each fold selected once as the validation subset (see Figure 2.2). The final validation metric is calculated by averaging its obtained value over the k folds. A particular case of k -fold cross-validation is Leave-One-Out (LOO) cross-validation, where each validation fold consists of only one data point, and the cross-validation loop is run n times, n being the number of data points.

Finally, the best model obtained during validation phase (with tuned hyperparameters) can be formally evaluated on a test set, leading to a testing error. A test set is an independent subset of data that the model has never seen during training or validation phases. This practice, called *model testing* or *external validation*, ensures that the performance assessment reflects the model’s capabilities in real-world scenarios. Best practices for selecting an independent test set include ensuring representativeness of the overall data distribution, maintaining similar ratios of classes in classification tasks, and confirming that the test set is sufficiently large to provide reliable performance estimates.

Importantly, in order for these approaches to yield accurate estimates of the model performance on new data, it must be ensured that when performing all aspects of model fitting, only the training observations are used [25].

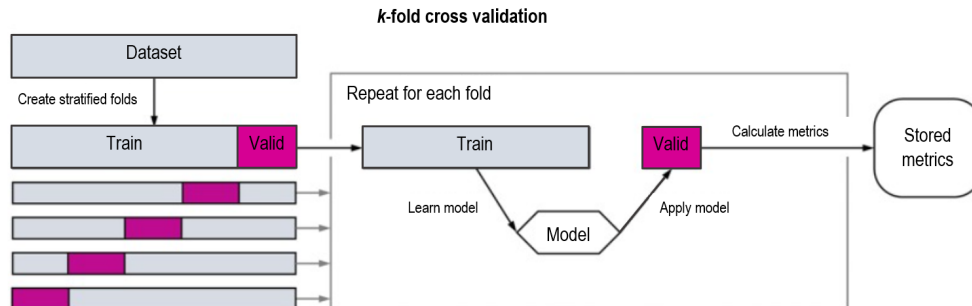


Figure 2.2: Schematic overview of k -fold cross-validation. The dataset is randomly split into k stratified folds. Each fold is used as a validation set once, while the other folds are temporarily combined to form a training set for training one model. Performance metrics on the validation set are calculated and stored, and the process is repeated for the number of folds k . The final validation metric is calculated by averaging its obtained value over the k folds. Figure adapted from [30].

This may seem obvious, but in practice, it is easy to create a *data leakage*, i.e. an un-intended contamination of training data by validation or testing data, causing optimistic bias in model performance evaluation [31]. We will come back on this tricky problem in Chapter 10.

To conclude, let us make a final comment on the terms ‘validation’ and ‘testing’ used throughout this section. The use of these terms might seem counter-intuitive at first, especially because of how the same terms are used in other contexts. Indeed, on the one hand, the ‘validation set’ is here used as part of the training process (in the broader sense), to enable model selection and hyperparameter tuning. This looks like internal testing, for example in software engineering contexts. On the other hand, the ‘test set’ is here used for final unbiased model evaluation, which sounds very much like ‘model validation’ in other contexts, e.g. when discussing ‘biological validation’ of a model. This confusing terminology is a bit unfortunate, but it however corresponds to the widely established ML one.

2.6 Model evaluation metrics

This section is devoted to the definition of popular model evaluation metrics. Note that these metrics can then be applied in training, validation or testing contexts.

n = 100	Actual: +	Actual: -
Predicted: +	TP = 59	FP = 8
Predicted: -	FN = 6	TN = 27

Figure 2.3: Example of a confusion matrix, for $n = 100$. Green elements of the matrix represent correct model predictions, while red elements refer to incorrect model predictions.

2.6.1 Classification metrics

Classification metrics are here introduced in the context of binary outcomes, for which we need to define which class level is, by convention, considered as the *positive* outcome, and which is the *negative* one. However, these metrics can be generalised to multiclass classification by specifying per-class metrics. In this latter case, we then define, for a given class, a binary outcome which contrasts the data point having this class as outcome, which is then positive by convention, vs. all the other data points, having a negative outcome.

To define these metrics, we need to introduce some preliminary indicators. Let TP be the number of *true positives*, defined as the number of correct positive model predictions, i.e. when both the prediction and the corresponding true value are positive. Let FP be the number of *false positives*, counting the number of cases where the model predicts a positive outcome, but the true value is negative. Similarly, let TN and FN be the number of *true negatives* and *false negatives*. These four indicators are the four elements of the 2×2 *confusion matrix*. They sum up to n , the number of data points. An example of a confusion matrix, with $n = 100$, is given in Figure 2.3.

We can then define the following binary classification metrics :

- the *accuracy* (*acc*) is the proportion of correctly classified data points (both positive or negative):

$$\text{acc} = \frac{\text{TP} + \text{TN}}{n} \quad (2.1)$$

- the *sensitivity* (*sensi*) or *recall* is the proportion of positive data points that are predicted correctly:

n (total population)	Actual: +	Actual: -	
Predicted: +	TP <i>True Positive</i>	FP <i>False Positive</i>	precision = TP / (TP+FP)
Predicted: -	FN <i>False Negative</i>	TN <i>True Negative</i>	Negative Predicted Value = TN / (FN+TN)
	sensitivity = TP / (TP+FN)	specificity = TN / (FP+TN)	accuracy = (TP + TN) / n

Figure 2.4: Overview of the main binary classification metrics, and their relation to corresponding confusion matrix elements.

$$\text{sensi} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.2)$$

- the *specificity* (*speci*) is the proportion of negative data points that are predicted correctly:

$$\text{speci} = \frac{\text{TN}}{\text{TN} + \text{FP}} \quad (2.3)$$

- the *precision* (*preci*), or *Positive Predictive Value* (*PPV*) is the proportion of positive predictions that are correct:

$$\text{preci} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2.4)$$

- the *Negative Predictive Value* (*NPV*) is the proportion of negative predictions that are correct:

$$\text{NPV} = \frac{\text{TN}}{\text{TN} + \text{FN}} \quad (2.5)$$

Figure 2.4 provides an overview of the metrics introduced above, illustrating how each relates to specific elements of the confusion matrix.

Some other metrics are specifically designed to address imbalanced classification problems, i.e. where the dataset has a strong imbalance between class occurrences:

- the *Balanced Accuracy* (*BAcc*) is the arithmetic mean of *sensi* and *speci*:

$$\text{BAcc} = 0.5(\text{sensi} + \text{speci}) \quad (2.6)$$

- the *F1-score* (*F1*) is the harmonic mean of *preci* and *sensi*:

$$\text{F1} = \frac{1}{\frac{1}{\text{preci}} + \frac{1}{\text{sensi}}} \quad (2.7)$$

Importantly, all the classification metrics introduced here have their values between 0 and 1, with a perfect classifier leading to a value of 1, and a random classifier leading to a value of 0.5 on average.

ROC curve

A key concept associated with binary classification metrics is the Receiver Operating Characteristic (ROC) curve [32]. Often, the ML model prediction for a binary classification consists in a predicted probability of positive outcome. In that case, a threshold, or *cut-off value* still needs to be set as a decision criterion for classification. While 0.5 seems an intuitive value for the cut-off value, other choices could be sensible depending on the context, and will have the effect of balancing between *sensi* and *speci* of the model. By letting the cut-off value vary between 0 and 1, different values of these two metrics are obtained, and can be represented on a plot. The ROC curve is such a plot, having *sensi* represented on the y axis, and $(1-\text{speci})$ represented on the x-axis (Figure 2.5). The ROC shows the various compromises between *sensi* and *speci* that can be obtained from the model. The area between this curve and the x axis defines the Area Under the Curve (AUC), which is another useful classification metric. The AUC can be seen as the intrinsic quality of a binary classification model to discriminate between the two possible class outcomes. Its value lies between 0 and 1, with 0.5 corresponding to a random classifier, having no discriminating power.

In multi-class classification, ROC curves and AUC values are typically generalised using one of three main strategies. In the one-vs-rest approach, a separate ROC curve is computed for each class against all others, allowing per-class and averaged AUC values. The one-vs-one approach instead compares each pair of classes individually and averages the resulting pairwise AUCs [34]. Finally, micro and macro averaging methods calculate a global ROC curve by either summing confusion matrix elements - TP, FP, TN, FN - across all classes (micro) or averaging per-class *sensi*'s et *speci*'s equally (macro).

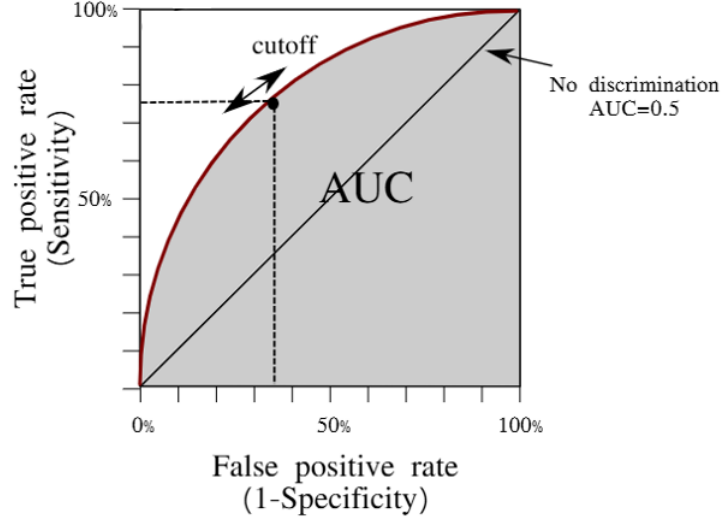


Figure 2.5: ROC curve and AUC. Figure adapted from [33].

2.6.2 Regression metrics

To define regression metrics, let us first introduce a few mathematical notations. In what follows, n is the number of data points, Y_i ($i : 1, \dots, n$) is the true outcome for data point i , and $\hat{Y}_i$ is the corresponding model prediction. $\bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i$ is the average of the true outcome values for all data points.

Some popular metrics used in regression problems are the following:

2.6.2.1 MSE and RMSE

The Mean Squared Error (MSE) measures the average of the squared differences between predicted and observed values:

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2 \quad (2.8)$$

The Root Mean Squared Error (RMSE) is the square root of the MSE and represents the average prediction error in the same units as the outcome:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2} \quad (2.9)$$

[MSE](#) and [RMSE](#) are the most popular metrics. Due to the squared error, they tend to further penalise larger prediction errors. Therefore, they are also highly sensitive to outliers, i.e. aberrant data points. The [RMSE](#) has the advantage of being expressed in the same units as the outcome, which facilitates interpretation.

2.6.2.2 MAE

The Mean Absolute Error ([MAE](#)) measures the average absolute difference between predicted and observed values, providing a more robust metric against outliers:

$$MAE = \frac{1}{n} \sum_{i=1}^n |\hat{Y}_i - Y_i| \quad (2.10)$$

2.6.2.3 R^2 (R squared, or coefficient of determination)

The coefficient of determination (R^2) quantifies the proportion of the variance in the observed data that is explained by the model:

$$R^2 = 1 - \frac{\sum_{i=1}^n (\hat{Y}_i - Y_i)^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2} \quad (2.11)$$

2.6.2.4 Concordance Correlation Coefficient

The Concordance Correlation Coefficient ([CCC](#)) [\[35\]](#) is similar to a Pearson correlation [\[36\]](#), with as main difference that it also accounts for location and scale shifts between the prediction and the true outcome. As with the Pearson correlation, [CCC](#) values lie in the $[-1, 1]$ interval, with good models having a [CCC](#) approaching 1. Detailed mathematical formulas, both for the [CCC](#) and Pearson correlation, can be found in [Appendix A](#).

2.6.3 Survival analysis metrics

As mentioned in [Section 2.4](#), survival data are unique in that some of the observations are censored. This means that for these data points, only a lower bound of the time to event is actually observed. This data characteristic implies that the popular regression metrics mentioned above cannot be directly used in the survival analysis case. Indeed, the intuitive transposition of these metrics to survival data, by using complete cases only, leads to suboptimal use of the information contained in censored observations, and to biased performance estimates [\[37\]](#). Therefore, in order to estimate survival data model

prediction accuracy, one needs to resort to other metrics, i.e. *concordance indices*, which compare the ranks of the predictions with respect to those of the true outcomes. In what follows, we begin by introducing some definitions and mathematical notations relevant to survival data, and then define concordance indices in the subsequent subsection.

2.6.3.1 Definitions and mathematical notations

Time to event and censorship

Let δ_i be the *observational outcome status* of data point i , i.e. $\delta_i = 1$ if the i th time to event has been observed, and $\delta_i = 0$ if the i th observation is censored. Let t_i be the recorded time for data point i , either the true time to event, or time of censorship.

Comparable vs. non comparable pairs of data points

A pair of survival outcome data points (i, j) is *comparable* if and only if

- $\delta_i = 1$ and $\delta_j = 1$ (none of the two observations are censored), or,
- $\delta_i = 1, \delta_j = 0$ and $t_i < t_j$ (only one of the two observations is censored, and its censor time is superior to the observed event time of the other paired data point).

In the other possible cases - both observations are censored, or the censor time of one observation is inferior to the observed time of the paired data point - there is not enough available information to compare the true times to event of the data point pair, and the pair is therefore non comparable.

Concordant vs. non concordant pairs of data points

Let us consider a survival outcome prediction model, and let $\hat{t}_i$ be its predicted time to event for data point i , with $i : 1, \dots, n$. A comparable pair of survival outcome data points (i, j) is *concordant* (given the model predictions), if and only if:

- if $t_i < t_j$ then $\hat{t}_i < \hat{t}_j$
- if $t_i > t_j$ then $\hat{t}_i > \hat{t}_j$

Similarly, a comparable pair of survival outcome data points (i, j) is *discordant* according (given the model predictions), if and only if:

- if $t_i < t_j$ then $\hat{t}_i > \hat{t}_j$
- if $t_i > t_j$ then $\hat{t}_i < \hat{t}_j$

Finally, considering all the possible pairs (t_i, t_j) , let us also designate by N^+ , the number of concordant pairs, N^- , the number of discordant pairs, and $N^=$, the number of ties, i.e. data point pairs for which $\hat{t}_i = \hat{t}_j$, or $t_i = t_j$, or both.

2.6.3.2 Concordance Index (**C-index**)

The **C-index**, also called C-statistic [37], quantifies the consistency between predicted and observed survival times, by comparing the ranks of the model predictions with respect to the true times to event. The **C-index** is formally defined as the probability that a comparable pair of data points is concordant for the particular predictive model at hand. Using the definitions of previous paragraphs, one can show [38] that the **C-index** can be estimated as follows :

$$\text{C-index} = \frac{N^+ + 0.5N^=}{N^+ + N^- + N^=} \quad (2.12)$$

C-index interpretation

The **C-index** assesses a model's discriminatory power, i.e. its ability to differentiate between individuals with different survival outcomes. A higher value indicates better discrimination and more accurate predictions. The **C-index** is intuitive and varies between 0 (completely discordant model predictions) and 1 (perfectly concordant model predictions), 0.5 representing a random prediction model. This makes it a suitable metric to compare different candidate models for a given dataset.

However, its absolute value can be misleading as it is essentially non linear in the $[0.5;1.0]$ range. Figure 2.6 shows, for a simplified case where there is no censorship, an intrinsic relationship between the value of **C-index** and the percentage of data point sets for which the predictions are 'incorrect' - here to be understood as 'at random', as opposed to 'properly capturing the event risk differences between subjects' (see [39] for a more precise definition). It suggests that obtaining a decent percentage of 'correctly assigned' event risks requires values of **C-index** that are close to one.

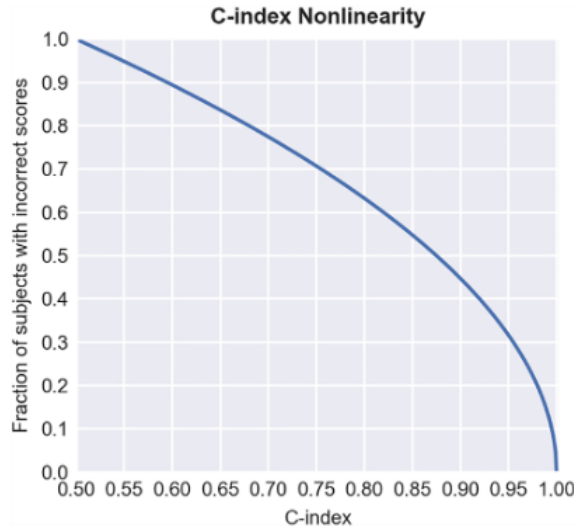


Figure 2.6: Graphical representation of the non-linear relationship between the [C-index](#) and the proportion of subjects with incorrectly assigned event risk scores. Figure reproduced from [39].

2.6.3.3 Unbiased version of [C-index](#)

The [C-index](#), as defined in Equation 2.12, still has a residual bias, in that it tends to overestimate the true [C-index](#), especially when the proportion of censored events is high. This is shown in [40], where the authors therefore proposed an unbiased version of the [C-index](#). The Inverse Probability of Censoring Weighted (IPCW) [C-index](#) consists in using Equation 2.12, but with slightly amended versions of N^+ , N^- and $N^=$ where each data point contribution is weighted according to its inverse probability of being censored. The latter probability can be estimated using Kaplan-Meier estimator [41] of the censoring curve.

Chapter 3

Introduction to Convolutional Neural Network models

In this chapter, we introduce a specific class of [ML](#) models, i.e. Convolutional Neural Network ([CNN](#)) models. Our motivation lies in the use of these models for identifying bio-markers of clinical outcomes in cytometry data, which will be the aim of [Chapter 10](#).

3.1 Artificial Neural Networks

3.1.1 Artificial neurons

[ANNs](#) are a class of [ML](#) models, which are inspired by the neuronal connections in the human brain. They are composed of artificial neurons, of which the mathematical model [\[42\]](#) is represented in [Figure 3.1](#).

A neuron receives one or more inputs, each assigned a weight that reflects its importance in solving the given problem. Depending on the layer in which the neuron is located (see below), these inputs may come from other neurons or directly from the input dataset. Given a set of inputs $x_j, j = 1, \dots, n$, the neuron computes its output y according to [Equation 3.1](#):

$$y = f(w_0 + \sum_{j=1}^n w_j x_j) \quad (3.1)$$

where w_0 is the bias, w_j is the weight associated to input x_j and f is the activation function (see [Section 3.1.2](#)).

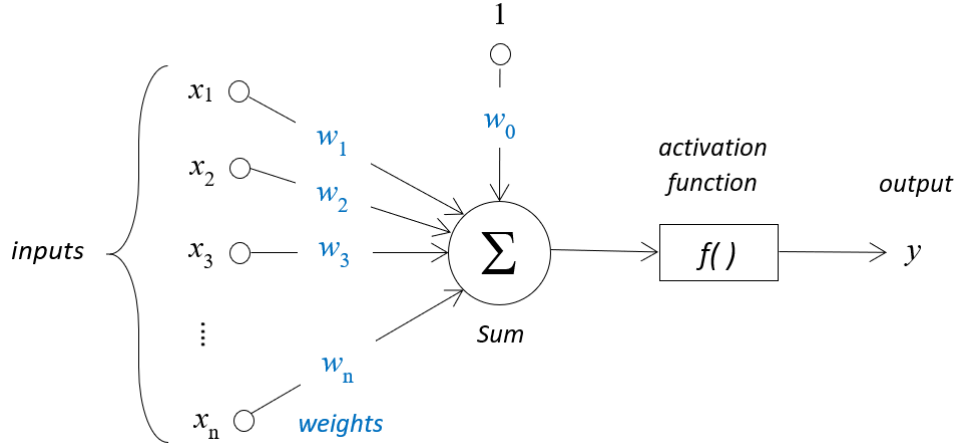


Figure 3.1: Mathematical model of an ANN.

Artificial neuron units are connected to each other and are arranged in layers (Figure 3.2). The first layer, which receives its input from the (external) input dataset, is called the input layer. The output layer is the last layer, which provides the final output of the neural network model. All the intermediate outputs are called hidden layers. A hidden layer receives its input from another layer of neurons, and provides its output to the next layer. Neural networks with fully connected layers, like the one shown in Figure 3.2, are called Multi Layer Perceptrons (MLPs) [43], and are the historical network architectures, at the basis of all modern ANNs. Networks with only a few hidden layers are called *shallow* networks, while networks with numerous hidden layers are called *deep* networks, a term which gave rise to the Deep Learning (DL) domain. By convention, networks that contains 1 to 2 hidden layers are usually considered as shallow, while deep networks have typically more than 3 hidden layers, and modern architectures can have dozens, hundreds, or even thousands of layers, depending on the application and architecture type.

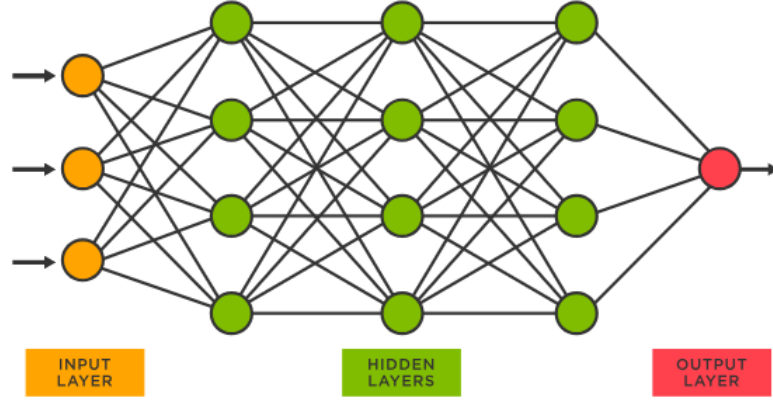


Figure 3.2: Organization of ANNs in layers.

3.1.2 Activation functions

The activation function, which transforms the weighted input sum, is key in the artificial neuron mathematical model (Figure 3.1), as it is the only element introducing non linearity in the model, allowing the ANN to learn complex non linear relationships between the inputs and output. The mostly used activation functions are the following (plots on Figure 3.3):

Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x) \quad (3.2)$$

Sigmoid

$$f(x) = \frac{1}{1 + \exp(-x)} \quad (3.3)$$

Hyperbolic Tangent (tanh)

$$f(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} \quad (3.4)$$

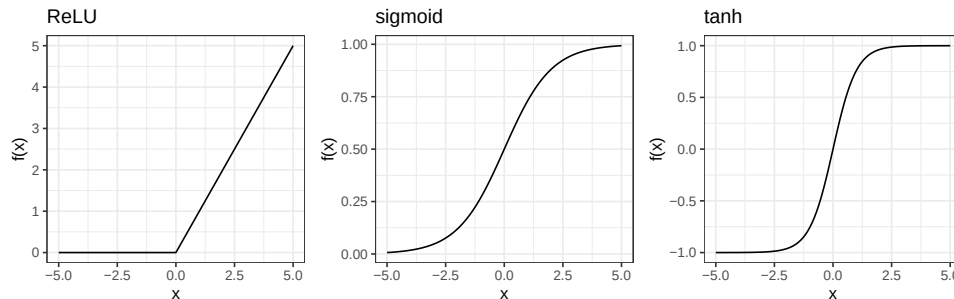


Figure 3.3: Some popular activation functions.

The sigmoid function was widely used as an activation function in the early development of ANNs, largely because its output values lie between 0 and 1 - an appealing feature in the biological analogy of neurons switching ‘on’ or ‘off’. However, nowadays the use of the sigmoid activation is mostly restricted to the output layer of binary classification models, where it naturally represents probabilities between 0 and 1. This shift occurred because the sigmoid function tends to saturate for strongly positive or negative input values, leading to the so-called vanishing gradient problem [44]. This issue severely limits the learning capacity of deep networks.

In contrast, the ReLU activation function effectively mitigates vanishing gradients and has become the default choice in most modern architectures - except for the output layer, which needs to be suited to the output value range. The tanh function remains useful in some cases - particularly in smaller or recurrent networks with normalised inputs - thanks to its symmetry around zero, which promotes more balanced gradient propagation. Note that ReLU functions are not differentiable at (exactly) zero, which requires attention when differentiating the loss function w.r.t. to the network weights - see section 3.2.1.

3.2 Network training

Network training is the iterative process through which the parameters of the model, i.e. the network weights, are progressively updated to fit the training dataset. In each iteration, or *epoch*¹, a number of training samples are presented as input to the network, which generates new predictions (using Equation 3.1 at each node). Based on a suitable error metric, i.e. the *loss function*

¹In ML, epoch is the term used to designate a model training iteration

or *cost function*, the observed discrepancies between network predictions and the corresponding true outcome values are calculated. Then a calculation called *backpropagation* is used to evaluate the sensitivity (or mathematically, the *gradient*) of the error with respect to each of the network weights. These weights are then updated through an optimisation algorithm, which aims at selecting updated weight values that tend to decrease the value of the loss function. These algorithms are variants of the gradient descent algorithm (see Section 3.2.3.1).

ANN training therefore needs three ingredients: a suitable loss function - given the type of outcome at hand, a backpropagation mechanism, and an optimisation algorithm which selects updated weights at the end of each epoch [45].

3.2.1 Loss functions

A suitable loss function must have two characteristics:

- it should be continuous and differentiable - with respect to network weights - across its whole domain of definition. This is required for using a gradient descent type algorithm.
- it must decrease when the prediction error decreases; as a consequence, different loss functions need to be used for different types of outcome, as these lead to different error metrics (see Section 2.6).

3.2.1.1 Loss function for regression problems

In case of a continuous outcome variable, a natural choice for the loss function is the [MSE](#), as defined in Equation 2.8, or its square root version, the [RMSE](#) (Equation 2.9).

3.2.1.2 Loss function for binary classification problems

In case of a binary categorical outcome variable, since the error metrics defined in Section 2.6.1 typically present jumps and are therefore not differentiable with respect to network weights, a popular choice of loss function is the binary cross-entropy. Using analogous mathematical notations as in Section 2.6.2, it is defined as follows [26]:

$$\text{Binary Cross-Entropy} = -\frac{1}{n} \sum_{i=1}^n (y_i * \ln(\hat{y}_i) + (1 - y_i) * \ln(1 - \hat{y}_i)) \quad (3.5)$$

where $\hat{y}_i$ is here the probability of positive outcome for data point i , as predicted by the model.

3.2.1.3 Loss function for survival regression problems

As discussed in Section 2.6.3, evaluating ML models for survival outcomes is trickier, and unfortunately, the C-index type of error metrics are not differentiable, and therefore not suitable as a loss function.

In practice, when applying ANNs to predicting survival outcomes, the output of the network is not directly the time to event, but a *risk score* assigned to each individual sample [46, 47, 48]. Risk scores are defined by analogy with a Cox Proportional Hazards (CPH) model, a statistical survival linear regression model that is widely used in survival data analysis [49]. Risk scores are continuous variables quantifying the risk of occurrence of the event for a given individual, in a given time interval, in comparison to a baseline risk. As such, they are useful to assess the relative risks of event occurrence for different individuals. However, translating risk score predictions into time to event predictions is not straightforward, as it requires the specification of a baseline hazard curve [50].

With risk scores as ANN output, the loss function used for survival problems is the *negative partial log-likelihood* of the CPH model [46]. Its mathematical expression is shown in Equation 3.6:

$$l(\hat{r}_1, \dots, \hat{r}_n) = - \sum_{i=1}^n \delta_i [\hat{r}_i - \ln(\sum_{j \in R_i} \exp(\hat{r}_j))] \quad (3.6)$$

where $\delta_i = 1$ if the event has been observed for sample i , and $\delta_i = 0$ otherwise, $\hat{r}_i$ is the model predicted risk score of sample i , and R_i is the *risk set* of sample i , i.e. the set of individuals for which the event has not yet occurred at time t_i .

The negative partial log-likelihood possesses the two needed characteristics of a loss function (Section 3.2.1), as it is continuous and differentiable, and it decreases when the predicted risk scores are more consistent with the observed or censored times. However, the latter is difficult to see from Equation 3.6. Indeed, the Right Hand Side (RHS) of Equation 3.6 is not a sum of error terms per individual sample, as is usually the case for the loss functions defined for classification and regression problems, where the contribution to

the error of each sample can be easily isolated. Rather, the [RHS](#) of Equation [3.6](#) is a complex, not intuitive expression of the risk scores, where the contribution of censored events is indirect through their belonging to the risk set of other samples. As a consequence, the value of this loss function depends, in an intricate way, on the number of observed events, the number of censored events, and the ordering of their event time or censorship time. This makes it challenging to interpret, especially for the comparison of model performance on different datasets, e.g. when comparing training loss with validation loss of a particular model. Therefore, to allow for a better interpretation and comparison of model evaluations, one needs to complement the loss function value with additional metrics that are easier to interpret, e.g. the [C-index](#) (Equation [2.12](#)).

3.2.2 Backpropagation

As previously noted, the purpose of backpropagation is to determine, at each training step, the direction in which each weight should be adjusted to minimise the overall error and improve the model predictions. In mathematical terms, this signal is the *gradient* of the loss function with respect to the network weights. While the gradient coefficients for the final layer weights can be computed directly, gradient coefficients corresponding to the inner layer connections need to be inferred iteratively by starting from the final layer, and working backward through the network, hence the term ‘backward propagation’².

3.2.3 Optimisation algorithms

This section briefly sketches some of the most popular optimisation algorithms used for performing the model parameter adjustment at each epoch. Its content is largely adapted from [\[45\]](#).

3.2.3.1 Gradient descent

The most straightforward way to leverage on the gradient of the loss function w.r.t. model weights, obtained via backpropagation, is to apply a *gradient descent*, or *steepest descent* algorithm. At each epoch, gradient descent decreases the value of the loss function by moving the weights in the direction of the negative gradient by a certain amount. This is done for all weight simultaneously. Formally, given $\mathbf{w}$, the vector of weights obtained at the start of

²Mathematically, what is done is applying the chain rule of derivatives recursively.

the current iteration, and the current gradient value $\Delta_w f(\mathbf{w})$, the algorithm updates the model weights to a new point $\mathbf{w}'$ such that:

$$\mathbf{w}' = \mathbf{w} - \epsilon \Delta_w f(\mathbf{w}) \quad (3.7)$$

where ϵ is the *learning rate*, a positive scalar determining the size of the step. The learning rate is a model hyperparameter that will require careful selection as part of model validation (see Section 2.5). Steepest descent converges when every element of the gradient is zero - or, in practice, very close to zero.

3.2.3.2 Stochastic gradient descent and minibatches

Gradient descent has to compute the full loss function in an epoch, including the contribution of each training data point. This can be problematic when the training set is large, as each weight update iteration can then be very slow. Stochastic Gradient Descent (SGD) algorithm is a variant of gradient descent, where only a small subset of randomly chosen data points are used at once to update the model weights. This small set of data points is called a *minibatch* and its size is another hyperparameter of the model. In that case, several minibatches, hence weight updates, are applied in one single epoch.

Importantly, SGD can be applied only if the loss function can be decomposed as a sum of each training set element contribution. As a consequence, SGD cannot be applied to survival analysis problems due to the complex form of the loss function (Section 3.2.1.3).

3.2.3.3 Adam algorithm

A widely used, improved version of gradient descent (or its stochastic counterpart) is the Adam algorithm [51]. Its popularity comes from its efficiency, and robustness to choices of its hyperparameters, making it straightforward to apply for various model fitting tasks.

Adam, which stands for *Adaptive Moment Estimation*, introduces two improvements to the (stochastic) gradient descent algorithm of previous sections (see [51] for the mathematical details) :

1. Momentum, which speeds up training by smoothing gradients steps by adding a fraction of the previous gradient to the current one.
2. Root Mean Square propagation (RMSProp), which looks at the steepness of the error surface for each parameter to update the learning rate

adaptively. Parameters with high gradients get smaller update steps, and low gradients allow bigger steps, avoiding to create oscillations for high gradient parameters.

These two concurrent mechanisms can be seen as applying a clever, refined version of the learning rate, which can adapt both as a function of previously seen gradients, and as a function of the relative gradient values for the different parameters.

3.3 Convolutional Neural Networks

3.3.1 Motivation

CNNs [52] are a specialized form of **ANNs** designed specifically for processing and interpreting visual data, such as images and videos. They have become the preferred architecture for computer vision tasks because of their ability to capture spatial structures and hierarchical patterns in images effectively. In contrast, fully connected **ANNs** architectures - like the one described in Section 3.1.1 - require a vast number of parameters in order to process such high dimensional image pixel inputs. This, in turn, leads to high computational costs and an increased risk of overfitting. **CNNs** mitigate these challenges by incorporating convolution operations, which enable more efficient feature extraction.

Importantly, the domain of application of **CNNs** is not restricted to two-dimensional (**2D**) image data. Ad hoc **CNN**-based architecture can accommodate various feature extraction tasks from data presenting specific spacial or time structure. As an example, **1D CNN** layers are frequently used for feature extraction from time-series data [53]. In Chapter 10, we will examine how **1D CNNs** can also be applied to cytometry data, where the inputs to the convolutional layers are unordered, in contrast to the ordered structure of time-series data.

3.3.2 Architecture overview

Figure 3.4 shows a typical **CNN** architecture for image classification. It represents an example of the famous *LeNet* family of models that was developed for one of the first real life applications of **CNNs**, i.e. hand-written character recognition [54]. In the following sections, we introduce some of the main building blocks (*layers*) that are usually included in a deep **CNN** model.

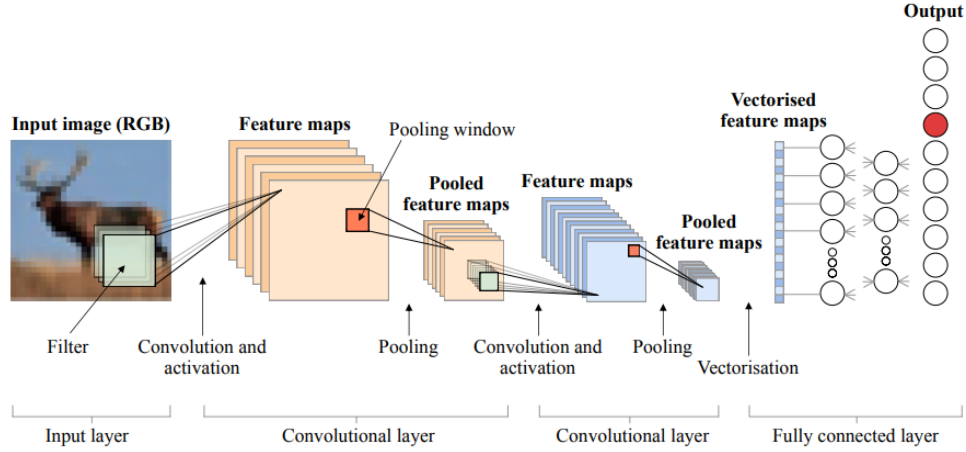


Figure 3.4: An example of LeNet architecture. The input is followed by two convolutional layers, and the output of the last convolutional layer is unrolled into a vector of activations and fed into a fully connected network. Finally the neuron with the largest activation in the output layer will correspond to the network prediction. Figure reproduced from [55].

3.3.2.1 Convolutional layers

The core elements of a **CNN** are *convolutional layers*. Unlike fully connected neurons, these layers use small filters - also called kernels or feature detectors - that move across the input. Each filter identifies distinct patterns or features, such as edges, corners, or textures (Figure 3.5).

In Figure 3.5, the input of the convolutional layer is a binary image represented as a rectangle of binary pixels. During the convolution operation, a *dot product* - i.e. an element-wise multiplication followed by a summation - is applied between the filter (kernel) weights, and a small region of pixels - called the filter *receptive field*. This dot product results in a single value of the output layer, which is called activation map or feature map. To obtain the full feature map, the filter needs to move through the whole input image, using the same set of filter weights, hence detecting the same feature in all small local region of the image. The convolutional layer has two key hyperparameters: the kernel size, which defines the dimensions of the filter, and the stride, which determines how far the filter moves with each step across the input. Moreover, in convolutional layers, several of these filters are applied, detecting different features from the image, hence creating several feature maps. The number of these feature maps is yet another hyperparameter of the convolutional layer.

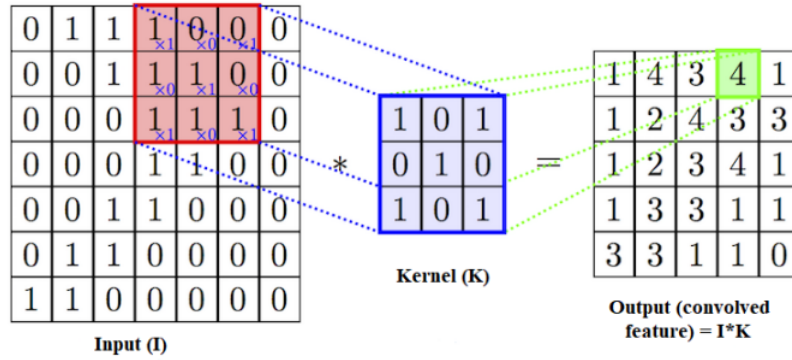


Figure 3.5: Details of a convolution operation. Figure reproduced from [56].

3.3.2.2 Pooling layers

Pooling layers, also known as downsampling or sub-sampling layers, reduce the input dimensionality by decreasing the number of parameters. Like convolutional layers, pooling layers apply a filter that moves across the input. However, unlike convolutional filters, pooling filters do not have weights. Instead, they apply an aggregation function to the values within their receptive field to generate the output. Widely used types of pooling are:

- max pooling: the filter selects the highest pixel value within its receptive field and pass it to the output.
- average pooling: the filter computes the average of the values in its receptive field and sends it to the output.

Although pooling layers discard some information, they provide significant advantages to CNNs, including reducing computational complexity, improving efficiency, and minimizing the risk of overfitting. Figure 3.6 illustrates the functioning of a max pooling operation.

3.3.2.3 Fully connected or dense layers

After multiple convolutional and pooling layers, CNNs typically include *fully connected layers*, also known as *dense layers*, to generate the final predictions, as illustrated in Figure 3.4. The output from the last pooling layer is first flattened into a 1D vector before being passed through fully connected layers. These layers link every single neuron from the previous layer, to all of those in the current layer, ultimately leading to the output layer, where predictions

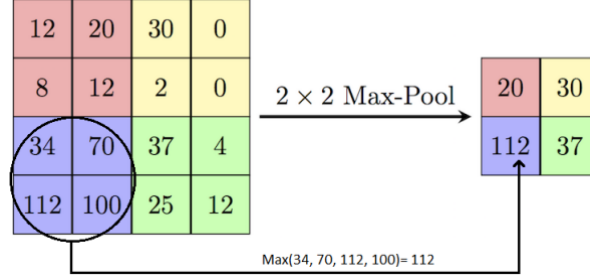


Figure 3.6: Details of a max pooling operation. Figure reproduced from [56].

are made. In fact, these dense layers are exactly the same as the traditional [MLP](#) architectures represented in Figure 3.2.

3.3.2.4 Batch normalisation layers

This section presents a relatively recently described (2015) architecture component, which is applicable for any type of deep [ANNs](#). It has been introduced for speeding up and stabilising the training process, and making it more robust to choices of initial weights and learning rate (see Section 3.2.3.1) [57]. Indeed, in deep neural networks, the input distribution for each layer can change during training, causing an issue known as *internal covariate shift*. Because of the typical shape of activation functions (Section 3.1.2), this shift makes it harder for subsequent layers to learn efficiently, as the input to some neurons can then be very low or very high. *Batch normalisation* helps mitigate this problem through the normalisation of activations in intermediate layers.

Formally, given a minibatch (see Section 3.2.3.2) of inputs: $x_1, x_2, \dots, x_m$, where m is the batch size, for a specific layer activation, the batch normalisation process involves the following steps [57]:

1. Compute the mean (μ) and variance (σ^2) of the activations across the batch:

$$\mu = \frac{1}{m} \sum_{i=1}^m x_i \quad (3.8)$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2 \quad (3.9)$$

2. Normalize the activations within the batch using the mean and variance:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad (3.10)$$

where ϵ is a small constant added to avoid division by zero.

3. Introduce learnable parameters γ (scale) and β (shift) to scale and shift the normalized activations, allowing the network to learn appropriate scaling and shifting:

$$y_i = \gamma \hat{x}_i + \beta \quad (3.11)$$

4. During training, the parameters γ and β are learned through backpropagation, just like other network parameters.

3.4 Dealing with overfitting

In this section, we investigate possible solutions to mitigate overfitting by deep neural networks. Overfitting has been defined in Section 2.2, in a generic ML model context. With their numerous weight parameters, deep ANN models are prone to overfitting, especially when the training dataset is scarce. One possible way to mitigate overfitting is to limit the complexity of the model, by e.g. decreasing the number of hidden layers or the number of neurons per layer. However, decreasing the complexity too much can also restrict the range of possible predictors, potentially leading to underfitting. The following sections are devoted to alternative, widely applicable techniques to avoid overfitting, notably with deep ANNs [58, 59].

3.4.1 Early stopping

Probably the most straightforward technique to mitigate overfitting is *early stopping*, i.e. tracking the model validation error (Section 2.5) in parallel with the training error during training, and stopping training when the validation error stops decreasing and starts to increase. This is illustrated in Figure 3.7.

A slightly different approach consists in running the training process until the target number of epochs is reached, and keeping track of the best model encountered during training. In other words, when the current model weights lead to a better validation loss - or another chosen validation metric - the current best model, i.e. the model weights, is saved as the new best model.

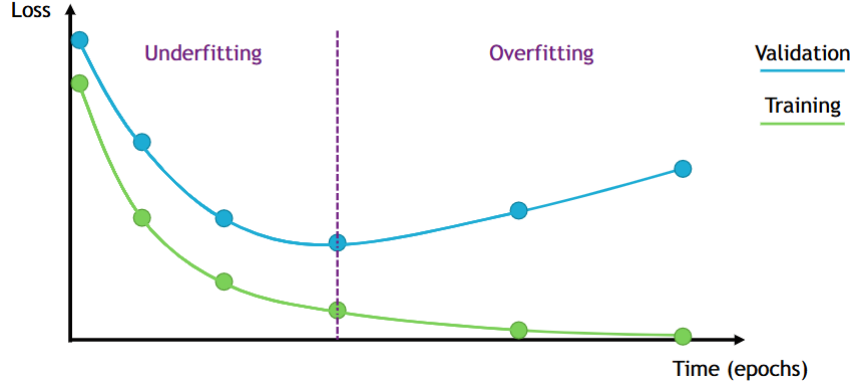


Figure 3.7: Illustration of the concept of early stopping. The model that should be selected corresponds to the dashed bar which is the point where the validation loss starts increasing. Figure reproduced from [60].

After training, the saved best model can then be restored and used for test and prediction. This method can prove useful when the validation error, seen as a function of the epoch, shows oscillations with a downward trend. In that case the early stopping method would stop too soon, while running the training process until the end, and keeping the best model, would avoid this pitfall. However, the downside of this approach is that a potentially large number of epochs could be run without improvement of the validation error, leading to unnecessary long run time of the training process.

3.4.2 Regularisation

In general, *regularisation* consists in penalising the dimensionality of the model, by adding a penalty term to the loss function. For ANNs, this penalty term targets the size of the network weights. Two main approaches exist, i.e. L1 or L2 regularisation.

In *L1 regularisation*, the loss function is augmented with terms proportional to the absolute values of the network weights:

$$\text{augmented loss} = \text{loss} + \lambda \sum_{i=1}^N |w_i| \quad (3.12)$$

where N is the total number of network weights (including bias).

This regularisation scheme is analogous to Lasso regression [61] in linear regression models, which tends to set coefficients of less influential covariates to zero, hence promoting sparser models. The λ parameter is a hyperparameter that can be chosen using model validation (Section 2.5).

In *L2 regularisation*, the loss function is augmented with terms proportional to the squared weights:

$$\text{augmented loss} = \text{loss} + \lambda \sum_{i=1}^N w_i^2 \quad (3.13)$$

This regularisation scheme is analogous to Ridge regression [62] in linear regression models, which tends to shrink the model coefficients towards zero, without forcing them to zero, hence promoting models with reduced covariate sensitivities.

3.4.3 Dropout layers

Dropout layers [63] help prevent overfitting by randomly dropping neurons (along with their connections) from the neural network during training. This prevents these neural units from co-adapting too much, hence forcing the network to generalise better. The dropout layer also includes a hyperparameter, i.e. the dropout rate or probability.

3.4.4 Data augmentation

Finally, *data augmentation* is yet another technique used with ANNs to address overfitting. It consists in augmenting the original training set with modified instances generated from the training data. Modifications can include introducing noise, applying geometric transformations (for image data), or other alterations [64]. The main objective is to present the model with slightly varied input and/or output data. This approach helps prevent the model from memorising overly specific patterns in the training data, improving its generalisation ability. However, applying data augmentation may require some creativity to design transformations that align with the specific characteristics of the inputs and outputs.

Chapter 4

Cytometry data analysis

4.1 Overview of data analysis workflow

This chapter is devoted to a description of the cytometry data analysis workflow. This workflow is schematically represented in Figure 4.1. It takes as input raw cytometry data generated via sample data acquisition - see Chapter 1, and is composed of a set of successive data treatments, visualisations, ML algorithms and statistical analyses, which are required to make sense of the collected cytometry data, and answer the scientific question of interest.

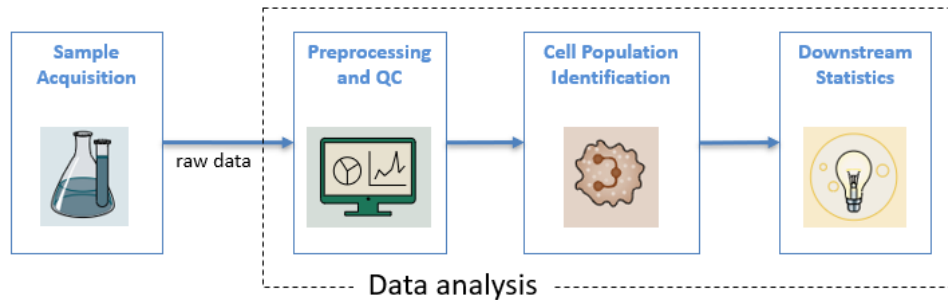


Figure 4.1: Schematic representation of the cytometry data analysis workflow (pictograms in text boxes reproduced from [65]). QC: quality control.

Such sets of successive data analysis steps, where the result of one data treatment serves as input for the next one, are often referred to as *bioinformatics pipelines*, or *data analysis pipelines*. Cytometry data analysis pipelines can usually be split into three main parts [65]:

- data preprocessing and Quality Control (QC), which consists in both

filtering undesirable and low quality events, and increasing the signal to noise ratio of the raw data, in order to feed the downstream steps with data of the highest possible quality (Sections 4.3 and 4.4);

- cell population identification, which aims at labelling the events with names of cell populations of interest (Section 4.5);
- downstream statistical analysis, which can range from the simplest descriptive statistics per cell population, to building complex prediction models for an outcome of interest (Section 4.6).

As will be discussed in the following sections, often there exist numerous ways of performing each data treatment step composing the boxes in Figure 4.1. In what follows, we will notably contrast *manual* methods with *automated* methods. Manual methods allow the user to perform the target data treatment manually and often visually, using a dedicated interactive software providing a Graphical User Interface (GUI). Automated methods perform data treatments using an automated algorithm, without relying on user intervention - except for algorithm parametrisation and configuration. For example, cell population identification is traditionally performed by *manual gating* - see Section 4.5.2 - but with the advent of high dimensional cytometry data, alternative automated algorithms are increasingly popular - see Section 4.5.3.

Finally, let us mention that some specific data analysis approaches can combine several boxes of Figure 4.1 at once. For instance, some ML models aim at predicting an outcome of interest, using the raw data - or minimally preprocessed raw data - directly as input, without going through intermediate steps. One example of such ‘all-in’ data analysis approach is discussed in Chapter 10.

4.2 Positioning of cytometry data among single-cell omic data types

In Chapter 1, we described the raw data structure obtained after data acquisition on a cytometer. Let us briefly recall that data pertaining to each acquired sample is stored in a data matrix, of which the rows represent the events (cells) and the columns represent the features (markers) - Figure 1.2. In fact, similar data structures - with each sample translating into a data matrix with features

omic data type	number of cells in one sample	number of features (gene/protein/marker)	obtained features
flow cytometry	100,000-10,000,000	10-30	targeted
mass cytometry	10,000-1,000,000	40-60	targeted
single-cell proteomics	100-1,000	thousands	untargeted
single-cell transcriptomics	1,000-10,000	tens of thousands	untargeted

Table 4.1: Data characteristics of some single-cell omic data types

in columns and cells in rows - can be found for several single-cell *omic*¹ data types, notably single-cell proteomics and single-cell transcriptomics. However, as shown in Table 4.1, cytometry data differ from other single-cell omics in that the number of cells to be analysed is typically orders of magnitude higher - millions vs. thousands or tens of thousands, while the number of features is much less - tens of features vs. tens of thousands for single-cell transcriptomics.

These similarities and differences between omic data types have important consequences in terms of the methods used for data analysis. For example, in single-cell transcriptomics, numerous methods exist for both batch effect correction (see Sections 4.3.6 and 4.3.7) and cell population identification. As a result of the data similarities, some of these methods can also be applied for cytometry data, but typically with some adjustments to take into account the high number of cells and lower number of markers in cytometry data.

Another important difference is that cytometry data analysis is *targeted*, i.e. it focuses on a limited number of pre-defined, known markers, of which the choice is driven by the biology and the scientific question of interest. In contrast, single-cell transcriptomic and proteomic data analysis are *untargeted*, i.e. they explore a broader range of genes or proteins, and the set of retained features depend on the experiment itself - e.g. some possible features may be eliminated due to the limit of detection of an instrument. As a result, cytometry data are generally used more often in hypothesis-driven research and less frequently in data-driven exploratory studies compared to their single-cell proteomics and transcriptomics counterparts.

¹The term *omic* refers to a set of disciplines (genomic, transcriptomic, proteomic, metabolomic), developed in the recent decades, that are aimed at the systematic and high-throughput analyses of biological samples at molecular level, with as underlying goal the better understanding of the functioning of living beings cells [66].

4.3 Preprocessing and quality control

In this section, we go deeper into the preprocessing and QC box of Figure 4.1. The overall objective of this data analysis part is to assess the quality and clean the data, as to obtain high quality data for the downstream analyses. This task can itself be split in several sub-tasks:

- compensation
- debarcoding in case of a multiplexed experiment
- scale transformation
- QC in time
- filtering of undersirable events like margins, debris, doublets and dead cells
- identification of outlier samples
- identification - and possibly correction - of *batch effects* (see Section 4.3.6 for a definition of batch effects)

In the next sections, we address these steps in more details.

4.3.1 Compensation

As mentioned in Section 1.4, in flow cytometry, compensation is a data preprocessing that consists in correcting for the spectral overlap of fluorochrome light emission. It requires the use of single stained controls, i.e. samples that are stained with only one fluorochrome at a time. More precisely, each single stained control should have both a positive and a negative population (i.e. stained and unstained beads or cells) to accurately define the spill-over.

4.3.1.1 Compensation for conventional flow cytometry

For decades, the compensation algorithm used in conventional flow cytometry has been the one described in [67]. It is performed in two steps:

1. First, the *spill-over matrix* is obtained. To this end, for each fluorochrome i , a single stained positive population is compared to an unstained negative population, and the corresponding signal in each channel j is measured as the difference in Median Fluorescence Intensity (MFI) between the two populations, as shown in Figure 4.2. This allows

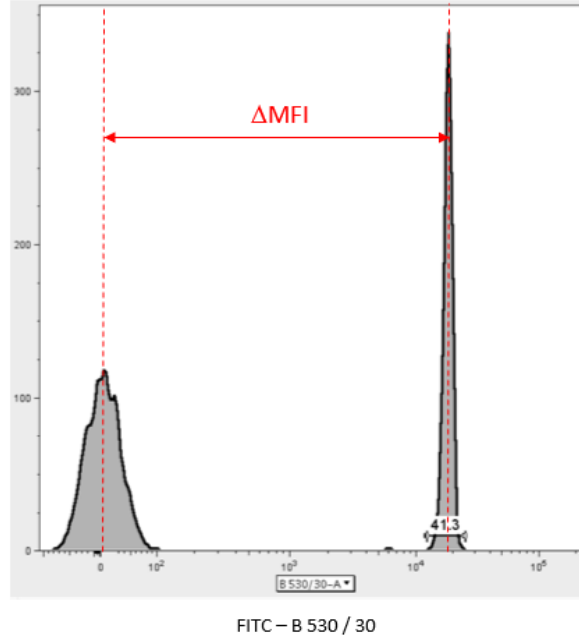


Figure 4.2: Measuring a single stained signal in a specific channel - here FITC (B 530/30) - by computing the difference of MFI between a positive (right peak) and a negative (left peak) population. Here the positive and negative populations are respectively single stained and unstained compensation beads.

to measure the spill-over of fluorochrome i into all channels $j : 1, \dots, n$, $i \neq j$, where n is the number of channels, as well as the useful signal in channel i . This set of values corresponds to one row of the spill-over matrix, and the same measurement is repeated for each fluorochrome-associated single stained sample.

2. Second, the compensation matrix is computed as the inverse of the spill-over matrix. This is done by the data acquisition or analysis software depending on the context. As a result, the compensation matrix can then be used to update the uncompensated event measurements, in order to obtain compensated values i.e. corrected for the spill-over.

Note that this method can, in theory, be extended in order to correct also for cell auto-fluorescence, by using an unstained subsample of these cells as a control sample, and using a dedicated channel to measure the auto-fluorescence signal [67]. However, this assumes that the auto-fluorescence is similar on all cells, which is not true in practice [68].

4.3.1.2 Alternative automated method - AutoSpill

The compensation method described above is relatively simple but has two drawbacks. First, it summarises the (positive) signal as a difference of MFI, which leads to a loss of information when averaging the intensities of the events for each of the two populations. Second, it requires the presence of clearly separated positive and negative populations, which might not always be the case for all markers if, for example, single stained cells - and not beads - are used as control samples.

To circumvent these issues, a new algorithm, *AutoSpill*, has been recently proposed [69]. The *Autospill* method is also based on single stained controls. However, it calculates the spill-over matrix without aggregating positive and negative populations into a single mean intensity, but instead uses robust linear regression, using each event as a data point, to assess the linear relationship between useful signal and spill-over of each fluorochrome. Importantly, this can only be done on the raw data intensities and not the *scaled* ones - see Section 4.3.3. This method offers the advantage of greater precision compared to the classical algorithm and remains effective even when positive and negative populations are not clearly separated. As a consequence, auto-fluorescence could also be more accurately corrected for, according to the authors [69].

4.3.1.3 Spectral unmixing for spectral flow cytometry

Compensation in spectral flow cytometry is conceptually very similar to conventional flow cytometry [70]. However, spectral flow cytometers make use of a larger number of detectors than fluorochromes. This makes the spill-over matrix calculated using single stained controls - see Section 4.3.1.1 - rectangular instead of square. As a consequence, the second step, i.e. calculation of the compensation matrix, cannot be performed using matrix inversion as the system of equation is overdetermined. Instead, statistical models like Ordinary Least Square (OLS) regression, or better, Poisson regression, can be resorted to [70].

The benefits of this ‘overdetermination’ are the following:

- additional detectors act as replicates for true signal measurement, allowing for better accuracy of compensated data
- discrimination between signal of fluorochromes with relatively similar emission spectra is possible
- it allows for cell type specific auto-fluorescence correction

In spectral flow cytometry, unlike in conventional flow cytometry, no single detector is designated as the ‘primary’ detector for a given fluorochrome, and the signals detected across all channels are not merely treated as ‘spillover’. Instead, all detectors are considered as providing equal value measurement for each signal source - either fluorochromes or cell auto-fluorescence, and the theory of ‘signal unmixing’ - well known and used in other signal processing contexts than cytometry - is applied in real time.

Let us note that the advantages of unmixing come with certain trade-offs. Compared to traditional compensation, unmixing depends more heavily on high-quality reference spectra, and carries a greater risk of error propagation across the dataset. Also, as it relies on statistical models, unmixing is more complex, hence it can be less intuitive and more challenging to troubleshoot.

4.3.1.4 Compensation for mass cytometry

As mass cytometry does not make use of fluorochromes having overlapping emission spectra, but rather heavy metal isotopes, the spill-over across channels, or *cross-talk* - is limited. The residual cross-talk can be due to the following causes [21]:

- isotopic impurity and natural abundance variations
- oxide formation during ionisation – having an impact of the isotope mass
- occurrence of doubly charged ions, having an impact on mass-over-charge (m/z) ratio
- abundance sensitivity, i.e. high abundance peak ‘bleeding’ into the adjacent m/z

Chevrier et al. [21] showed that this residual cross-talk was a linear function of true signal intensity (as in flow cytometry), and proposed a method for adjusting for it. This method is based on NNLS (Non Negative Least Squares) linear regression, which forces the estimated values to stay on the positive side, aiming at avoiding over-compensation.

4.3.2 Debarcoding for multiplexed experiment

Sample multiplexing is a process that can be used to reduce technical variability between acquired samples. It consists in pooling multiple biological samples together in a single run, using unique identifiers or *tags*, so that they can be

distinguished and analysed separately during data analysis. Multiplexing protocols exist both for flow and mass cytometry. However, sample multiplexing is more common in mass cytometry because of the following considerations:

- acquisition times are longer for mass cytometry, increasing the need to split sample acquisition into batches, and therefore worsening the technical variability between samples or batches.
- mass cytometry leads to minimal cross-talk between channels, allowing for more available channels than those in flow cytometry. Therefore, using some of these channels for tag identification is less costly.

Mass-tag cellular barcoding protocols are described e.g. in [71] and [72]. Before specific antibody staining, the cells are fixed, permeabilised, washed and labeled with a unique, sample specific combination of isotope masses to ‘barcode’ each sample. For example, this unique combination can be chosen according to a ‘*n-choose-k*’ barcoding scheme [72].

Data analysis of multiplexed samples therefore involves an additional preprocessing step, the *debarcoding*, which allows ‘back-allocation’ of each cell of the pooled sample to its original biological sample. For example, the single-cell deconvolution preprocessing described in [72] involves identifying the signal associated to each isotope included in the barcode, assessing positivity thresholds, and allocating each event from the pooled sample to either one of the original biological samples, or to a ‘to-be-removed’ category. Indeed, as a by-product, the ‘*n-choose-k*’ barcoding scheme allows to identify the doublets/multiplets combining cells of different original samples, as well as debris. Finally, at the end of the debarcoding, a separate, cleaned file for each original biological sample can be obtained. This debarcoding preprocessing method is implemented in e.g. the *CATALYST* Bioconductor R package [73].

4.3.3 Scale transformation

In cytometry, the variance of the measured intensity of a cell population tends to be proportional to its mean [74]. Therefore, a natural choice would be to use a logarithmic scale to compensate for this tendency by compressing data at the high end of the scale. This would allow populations to keep a similar shape on a log scale, whether they are expressing the marker at low or high intensity, although, in absolute terms, the high intensity populations exhibit a much greater standard deviation.

However, the simple logarithmic transformation has the drawback that it is not defined for zero and negative values, and creates visual artefacts for small values near zero. Nevertheless, these small (or even slightly negative) values are often met in practice, due to a number of technical reasons (automatic background corrections performed by hardware or software, variability of photon counting in detectors...). The problem worsens with compensated flow cytometry data, where the compensation spread - see Section 1.4 - can lead to negative values even relatively far from zero. Therefore, slightly more complex scale transformations, allowing for negative input values, and showing a ‘near linear’ behaviour around zero, are typically chosen to scale cytometry data. The most popular transformations are the *inverse hyperbolic sine* (*arcsinh*) transformation, and the *logicle* transformation. Both are monotonically increasing transformations, and are defined for the whole range of real values (including the negative domain).

The arcsinh transformation

The *arcsinh* transformation is the inverse of the hyperbolic sine (*sinh*) function, which is defined as follows:

$$f(x) = \frac{\exp(\frac{x}{cf}) - \exp(-\frac{x}{cf})}{2} \quad (4.1)$$

where *cf* is the *cofactor* parameter, with $cf > 0$.

It turns out that the inverse function, i.e. the *arcsinh*, also has an explicit analytical expression:

$$f(x) = \ln\left(\frac{x}{cf} + \sqrt{1 + \left(\frac{x}{cf}\right)^2}\right) \quad (4.2)$$

Figure 4.3 illustrates the *arcsinh* function for different cofactor values. In panel (a), the x-axis is shown on a linear scale, highlighting that the *arcsinh* transformation is approximately linear and symmetric around zero. In panel (b), the x-axis is displayed on a logarithmic scale, showing that the transformation behaves like a logarithmic function for large values. The cofactor parameter determines the location of the transition between the ‘linear’ and ‘logarithmic’ regions.

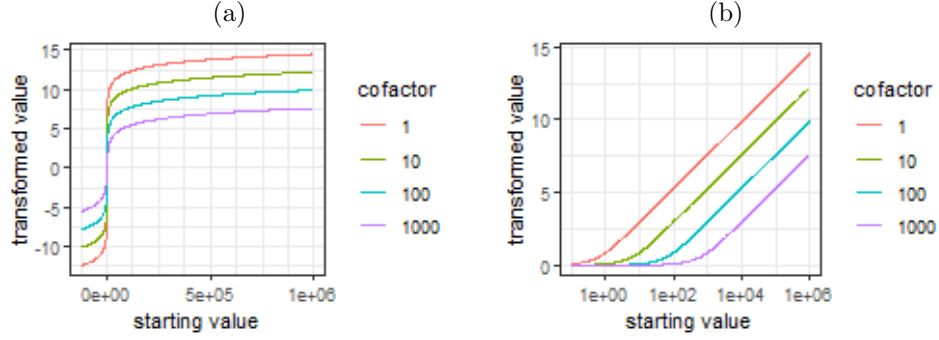


Figure 4.3: The inverse hyperbolic sine (arcsinh) transformation, for different values of the cofactor parameter. (a) x axis in linear scale; (b) x axis in log scale.

The logicle transformation

The *logicle* transformation has been first proposed for scale transformation of flow cytometry data in [75]. Practitioners sometimes refer to it as the *bi-exponential* transformation because its inverse is taken from the bi-exponential family of functions (Equation 4.3).

$$f(x) = a \exp(bx) - c \exp(-dx) + e \quad (4.3)$$

where a, b, c, d, e are parameters and $a, b, c, d > 0$.

By comparing this equation with Equation 4.1, one can see that the hyperbolic sine function is a particular case of bi-exponential function, where $a = c = 0.5$, $b = d = \frac{1}{cf}$, and $f = 0$. Now, the logicle transformation can be obtained by particularising Equation 4.3 as follows [76]:

1. restricting the parameter space in order to impose logarithmic behaviour in the high values, quasi-linear behaviour around zero, and smooth transition between these two domains, and
2. re-parametrising the function with 4 parameters, where 3 of them can be trivially assigned to scale characteristics (lower bound, upper bound, and number of decades to be displayed), so that the user is left with only one truly ‘free’ parameter, i.e. the *width*. This width parameter in the logicle transformation plays a similar role as the cofactor in the arcsinh transformation, i.e. it sets the extent of the domain around zero where the function has a (nearly) linear behaviour.

Adjusting the free parameter of the scale transformation

Setting an appropriate value for the free parameter of the scale transformation, i.e. the width in the logicle transformation, or the cofactor in the arcsinh transformation, can be done ‘manually’, by visual inspection, as is shown in Figure 4.4. The key idea is to select the value of the parameter such that a sample with a positive and a negative population - for the visualised channel - shows a nicely behaving bimodal distribution with well separated peaks [77]. Alternatively, an automated algorithm can also be used, and several methods have been proposed in the literature ([77], [78], [79], [80]).

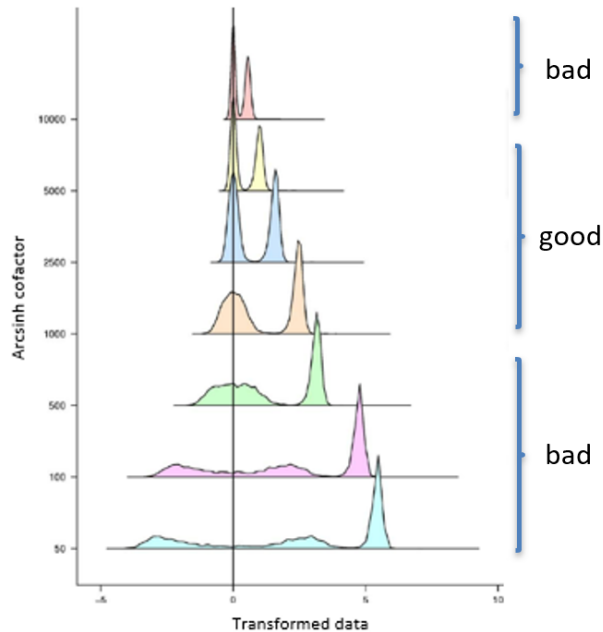


Figure 4.4: Setting the free parameter value of scale transformation manually, by visual inspection (here the cofactor of the arcsinh transformation). The goal is to increase the separation between the negative and positive populations without creating artefacts in the negative population. Too high values of the cofactor lead to insufficient separation, while too small values of the cofactor lead to artefacts in the negative population, with a third mode appearing in the overall distribution. Figure adapted from [77].

For mass cytometry data, the arcsinh transformation is the usual choice, as it is simpler, having only one parameter. However, for flow cytometry, the logicle transformation is more common, due to the fact that compensated data

typically show a wider spread into negative values. In that case, the logicle transformation provides more flexibility to accommodate well behaved transformed distributions in the negative and ‘close to zero’ domains.

To conclude, let us mention two important remarks. First, the need for an appropriate scale transformation is not only for data visualisation. For instance, having cell populations with adequately scaled intensity variance is very important for automated cell population discovery algorithm, e.g. clustering methods. Second, when the data analysis goal is to compare different cytometry samples of a study, it is key to apply the same scale transformation - i.e. with the same parameter values - to all samples, in order to avoid introducing artefacts, impacting the sample comparisons. In that case, one way of selecting the free parameter of the common scale transformation is to select it while using, as input, a synthetic sample, obtained by concatenating events from all samples of the study [81].

4.3.4 QC in time

General principle

The reliable acquisition of high-quality flow cytometry data depends on individual cells passing through the laser beam at a consistent flow rate, along with stable sample composition and constant optical and electronic instrument settings throughout the acquisition process. Under these conditions, signal stability is reflected by a uniform and consistent fluorescence intensity over time - that is, a stationary distribution - based on the assumption that particles randomly pass through the laser intercept at any given moment [4]. In contrast, signal instabilities can be due to various causes: clogging, variations in flow rate (voluntary or involuntary), gradual signal intensity change during startup, sample contamination, electronic drift... These instabilities are in turn responsible for technical artefacts in the data, which can disturb downstream data analyses [82].

In cytometry data analysis terminology, *QC* usually refers to the process of cleaning each sample, by removing the events that are parts of non stationary signal. However, in this work, we name this process ‘*QC in time*’, to emphasise the fact that it is only one of the *QC* checks that can be performed. For example, exploratory data visualisation in the sample dimension can be done to detect outlier samples and batch effects, which is another type of *QC* - see Section 4.4.

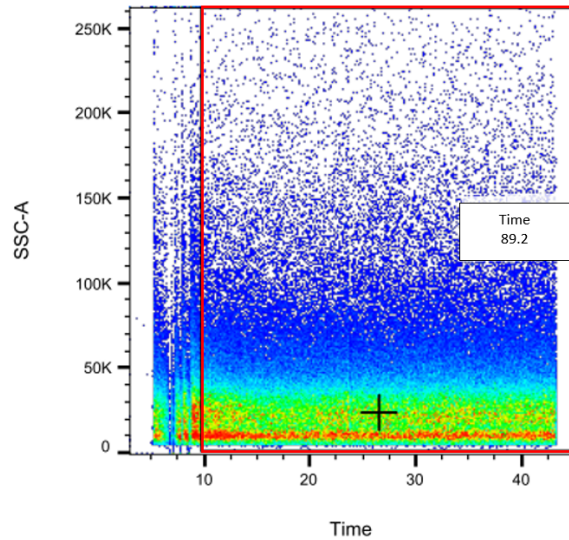


Figure 4.5: Applying a ‘Time’ gate to remove chunks of instable signal in time (here looking at the SSC-A channel). Kept events are inside the red rectangle.

‘QC in time’ can be done manually, by inspecting the signal distribution of selected channels, as a function of time. Parts that ‘look different from the rest’ are then removed using one or several rectangular time gate(s) - Figure 4.5. Removing an event translates into removing a full row of the data matrix, across all channels. In principle, this manual inspection should be done for every single channel, and for all samples. In practice however, often only one channel might be used.

Automated methods

The manual process of applying time gates to perform ‘QC in time’ of all samples of a study can be tedious and time consuming. Therefore automatic methods have also been developed. The most popular ones are listed below:

- *flowAI* [83] performs three different automatic checks: a flow rate stability check using outlier detection analysis, a signal measurement stability check for each channel using changepoint analysis of the medians, and a dynamic range check, again using outlier detection.
- *flowCut* [84] calculates summed density measures using the mean, median, several percentiles, skewness, and variation of the flow signal, per

time segment. It then removes aberrant events based on a density curve analysis.

- *PeacoQC* [82] performs signal summarisation by extracting the peaks in each channel. Then it filters outlying peaks using two types of methods: isolation trees - taking all markers into account at once - and per marker outlier detection. Besides, if a continuous drift is detected in one or several marker(s), a warning message is issued. In contrast with the other two methods, *PeacoQC* needs to be run on scale transformed data, not on raw data.

Bead-based normalisation for mass cytometry data

Bead-based normalisation can be considered as a form of ‘QC in time’, performed for mass cytometry data. Due to the lower data acquisition rates in mass cytometry, data acquisition of multiple samples might require splitting the samples in multiple batches and acquiring them on different days, which is prone to batch effects and instrument signal drift [85]. Therefore, in mass cytometry, bead-based normalisation is a key preprocessing step that corrects for signal intensity drift caused by instrumental fluctuations during long acquisition times. It involves spiking metal-labeled normalisation beads - containing stable, non-biological isotopes - into each sample. Since the bead composition is constant, any variation in their signal reflects technical noise rather than biological differences. During analysis, bead events are identified and used to model signal drift over time, typically by computing median intensities in short time bins. Correction factors are then calculated to scale all measured channels back to a consistent baseline. Such a normalisation method, based on an approach described in [86], is implemented e.g. in the *CATALYST* Bioconductor R package [73].

4.3.5 Removal of undesirable events

A series of ad hoc preprocessing steps are devoted to the filtering of undesirable events in cytometry sample data. This filtering of events translates into removing corresponding full rows of the data matrix - across all channels. These undesirable events are of different types: margin events, doublets, debris and dead cells.

4.3.5.1 Margin event removal

Margin events - also called boundary or edge events - are cells or particles whose measured signal lies at or beyond the limits of the instrument detection

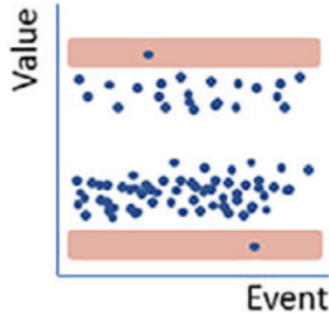


Figure 4.6: Removing margin events - figure reproduced from [82].

range, in one or more channel(s). These events are problematic because they do not reflect reliable, quantitative data and can distort downstream analyses. The causes for margin events are multiple: detector saturation due to too strong fluorescence signal (in flow cytometry), or too high concentrations of metal-labeled antibodies (in mass cytometry), low signal falling below detection threshold, poor instrument settings...

Margin events are filtered by manually or automatically gating them out, i.e. by removing the events with a channel intensity equal or lower than zero (for mass data or uncompensated flow data), or equal or higher than the maximum of the instrument - as schematised in Figure 4.6.

4.3.5.2 Doublet and multiplet removal

As described in Section 1.1.1, flow cytometry uses hydrodynamic focusing to align cells such that they can pass one by one through the laser beam. However, rare doublets (or even multiplets) can still occur, leading to artificially high signal intensities and misleading data, unless properly identified and removed. Traditional methods to remove doublets are based on gating, using specific combinations of channel signal. These combinations depend on the type of technology (flow or mass).

Doublet removal in flow cytometry

In a flow cytometer, as a particle passes through the laser, photons are converted to electric current by the PMT, and a voltage pulse is generated. As already mentioned in Section 1.1.3, the voltage pulse has three characteristics: area, height, and width - Figure 4.7 (a). The area is the overall amount

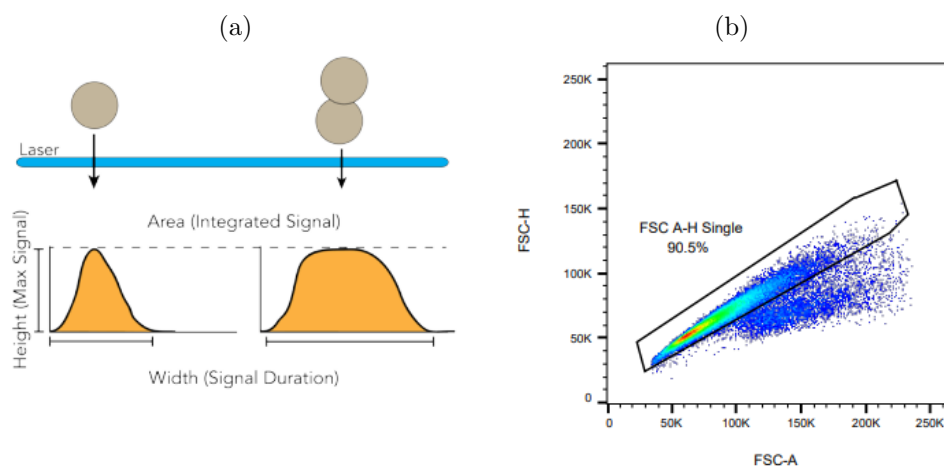


Figure 4.7: Doublet elimination in flow cytometry, using gating. (a) When an event corresponds to a doublet, it translates into a wider pulse, for a given signal height. (b) The FSC-H vs. FSC-A 2D view can be used to gate out doublets and multiplets - kept events are inside the polygon.

of fluorescence associated with an event, the height is the peak signal, and the width is the signal duration. Using two of those three characteristics related to one scattered light channel - FSC or SSC, can help identify single events associated with multiple particles, by either plotting width vs area, or plotting area vs height. On Figure 4.7 (b), an example of doublet elimination using gating is shown. Here, the events having a higher FSC-A to FSC-H ratio - located at the right at the main diagonal - are gated out.

On top of manual gating, automated doublet elimination methods are based on similar ideas. For example, doublet elimination methods are implemented, both in the *PeacoQC* [82], and in the *flowStats* [87] R packages. These methods first calculate the appropriate ratio - e.g. FSC-A to FSC-H - for each event, and then use outlier detection techniques to identify and eliminate doublets and multiplets.

Doublet removal in mass cytometry

Doublets elimination for mass cytometry is also mostly done using gating. Here, since there is no scattered light channels, DNA intercalator signals are used instead. A DNA intercalator is a molecule containing a specific metal isotope - e.g. ^{191}Ir or ^{193}Ir , which inserts itself between the base pairs of a DNA

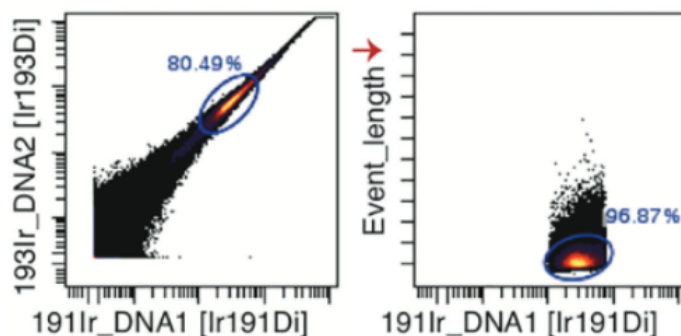


Figure 4.8: Doublet elimination in mass cytometry, using successive gatings in 193Ir vs. 191Ir 2D view, and in ‘event length’ vs. Ir191 2D view. Kept events are inside the ellipses. Figure reproduced from [20].

double helix without disrupting the covalent bonds between the nucleotides [20]. Then, the most common 2D views for manual gating are:

1. DNA intercalator vs. ‘event length’ (or TOF). Singlets generally have a lower event length and DNA signal, forming a distinct cluster, while doublets and aggregates appear as events with higher event length and increased DNA signal due to the presence of multiple nuclei or larger cell clusters.
2. DNA intercalator 1 (e.g. 191Ir) vs. DNA intercalator 2 (e.g. 193Ir). This plot typically shows a diagonal cluster for singlets and higher, off-diagonal clusters for doublets and aggregates.

Figure 4.8 shows an example of using these two gates successively.

***Cleanet*, a new method using all marker intensities**

The previously described methods for doublet elimination use proxies of cell size and DNA content, to try to identify multiple cell events. Recently, a new automated method, *Cleanet*, using the distribution intensities of all markers, has been proposed [88]. The idea behind *Cleanet* is to generate synthetic doublets in silico, by combining the intensities of randomly chosen single events from the sample being cleaned. Then, the original events that are too similar to the generated synthetic doublets are flagged as potential doublets. Note that this type of method can be used both for flow and mass cytometry.

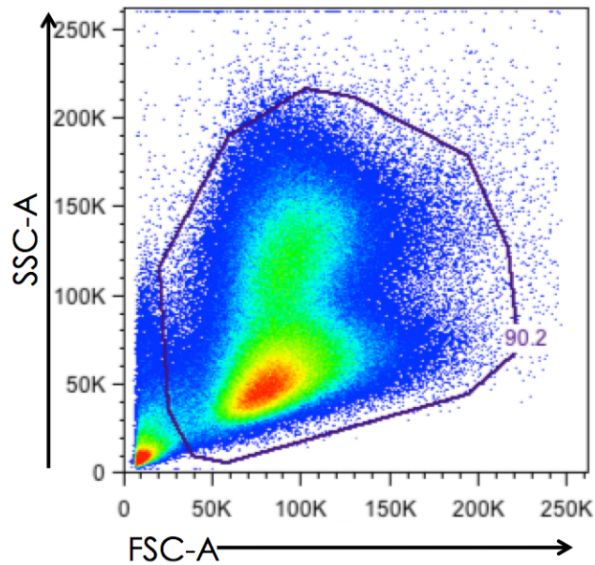


Figure 4.9: Example of debris removal gate, for flow cytometry data. Kept events are inside the polygon.

4.3.5.3 Debris removal

Debris are small particles that are not cells, and need to be eliminated, because they can generate false positives (e.g. dead cell fragments that can bind non specifically to antibodies). Since debris are of small size and have little internal content, it is natural to use corresponding channels to gate them out:

- for flow cytometry, use a **FSC** vs. **SSC 2D** view to gate out the cluster of events located nearest to the axes origin (see Figure 4.9).
- for mass cytometry, use an ‘event length’ vs. **DNA** intercalator view to gate out the cluster of events located nearest to the axes origin.

Note that there also exist automated methods to gate out debris. For instance, both *Cleanet* [88] and *Autospill* [69] provide such methods as part of their preprocessing.

4.3.5.4 Dead cells removal

Finally, dead cells also need to be removed. Indeed, because dead cells tend to bind non specifically to antibodies, they are an important source of false positive. The most reliable procedure to discriminate dead cells is to stain samples with a reagent staining dead cells specifically:

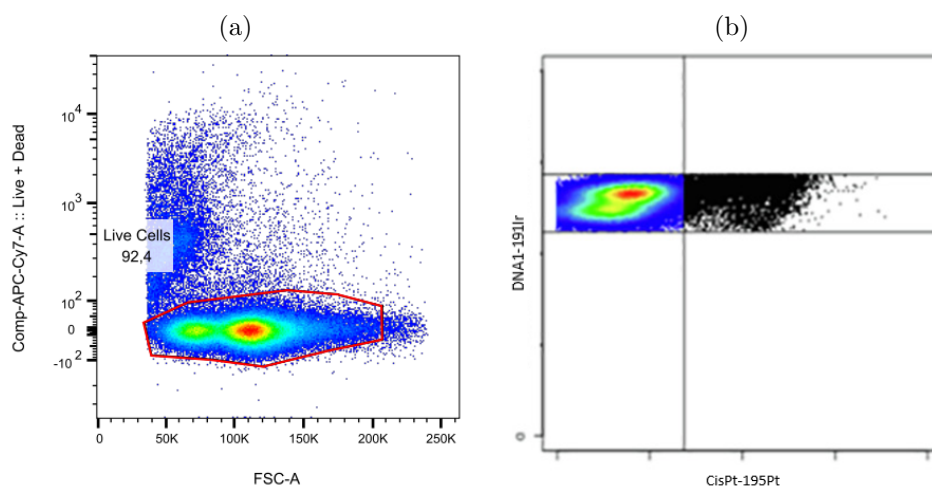


Figure 4.10: Dead cells elimination in cytometry, by manual gating. The gate consists in selecting a threshold on the intensities of a specific channel discriminating dead vs. alive cells. All events that are below the chosen threshold are then eliminated. (a) Flow cytometry example using ‘Live-Dead’ channel (kept events are inside the polygon); (b) Mass cytometry example using 195Pt Cisplatin channel (eliminated events are in black).

- amine-reactive dyes (LIVE/DEAD), or 4',6-diamidino-2-phenylindole (DAPI), or propidium iodide (PI) for flow cytometry samples
- 195Pt Cisplatin for mass cytometry samples

Eliminating dead cells is then a matter of correctly applying a threshold in the discriminating channel. This can be done manually - as shown in Figure 4.10, or automatically using packages such as *flowDensity* [89].

4.3.6 Batch effect and outlier sample detection

Batch effect is a generic term that designates technical variability between groups of samples. Possible sources of these technical variations can be numerous, especially for large data sets that are often generated over the course of days or weeks: differential antibody staining across samples, different batches of reagents, sequential sample handling and instrument drift, data acquisitions across different labs and different instruments in multicentric studies...

Exploratory data analysis can be performed to identify expected or unexpected batch effects. For instance, low dimensional projection at sample level -

method name	with/out reference sample	per cell population correction
<i>CytoNorm</i>	both(*)	yes
<i>cyCombine</i>	without	yes
<i>CytofBatchAdjust</i>	with	no
<i>Harmony</i>	without	yes
<i>CytofRUV</i>	with	yes

Table 4.2: Classification of some batch effect correction methods for cytometry data. (*) *CytoNorm* can also work with one batch aggregate, used as a proxy for a reference sample.

see Section 4.4 - can be used to visually identify clusters of samples that could be indicative of batch effects. Moreover, these plots are also helpful for spotting outlier samples, which might have specific quality issues. As such, this type of visual QC is complementary to the ‘QC in time’ methods described in Section 4.3.4. Data visualisation using low dimensional projection will be discussed in Section 4.4.

4.3.7 Batch effect correction

Batch effect correction methods attempt to correct for systematic differences across batches. This may be needed, as downstream analyses might be misled by the presence of batch effects in the data. Numerous methods exist, some of them developed specifically for cytometry data (*CytoNorm* [90], *cyCombine* [91], *CytofBatchAdjust* [92], *CytofRUV* [93],...) and others coming from the RNA sequencing data analysis realm (*Harmony* [94],...).

Batch effect correction methods can be classified according to two criteria (Table 4.2):

- whether they work using a reference sample, i.e. a technical replicate for each batch (*anchor*), or work with no reference sample, or can work in both settings;
- whether they calculate a common batch correction for all events, or, in contrast, calculate a specific correction per cell population. The latter is typically more difficult, since this implies preliminary clustering of the events into cell populations, which could itself be perturbed by the batch effects if they are too strong.

Batch correction methods need to be used with caution. If they ‘over-correct’, they might introduce artefacts in the data. This risk can be mitigated

by using diagnostic tools, e.g. plots enabling visual inspection of batch corrected marker distributions (*CytoBatchNorm* [95], *CytoNorm 2.0* [96]). Additionally, some dedicated control metrics can be calculated to make sure the method is effective in removing batch effects but keeps the biological signal alive [91]:

- the Earth Mover’s Distance (EMD) between batches should globally decrease (see the EMD definition in Section 4.4.4), indicating a decrease of batch effects in the data;
- the Median Absolute Deviation (MAD) of individual marker intensities within batches should approximately stay the same, indicating preservation of biological signal.

Finally, let us mention that batch effect correction is a particular case of dataset integration [91], where the markers are the same between the different datasets. For cytometry datasets, a more general - and complex - task is *panel integration* which also requires imputation of some markers that are not present in all datasets. See e.g. [97] for an interesting approach of panel integration using deep learning Variational Auto-Encoders (VAE).

4.3.8 Comment on the order of preprocessing steps

In the previous sections, we addressed numerous preprocessing steps: compensation, scale transformation, filtering of undesirable events, batch effect correction, ... All these steps are necessary to obtain, as much as possible, high quality data for downstream analyses. What is somehow less clear is the order in which these data treatments should be done. Actually this depends also on the methods chosen to perform each single step. For examples, it is well documented that specific preprocessing methods have to be performed on raw data (e.g. *flowAI* for ‘QC in time’), while others better work with compensated and/or scale transformed data (e.g. *PeacoQC*). But often, a suitable order of the needed preprocessing steps is not (clearly) documented. It stays that this order may have a noticeable impact on the end result, i.e. the events remaining in the cleaned, preprocessed data. Indeed, interaction between preprocessing steps is one source of complexity for building preprocessing pipelines - see Chapter 7.

4.4 Data visualisation using low dimensional projection

4.4.1 Introduction

In this section, we discuss low dimensional projection methods for visually representing cytometry data. Low dimensional projection, in the context of data visualisation, consists in transforming high dimensional data - data with many variables/features - into a space with fewer dimensions - typically two or three dimensions - so that humans can visualise and interpret patterns, clusters, or structures in the data. These visualisations are typically used as data exploratory analyses, i.e. for identifying or confirming patterns in the data. These patterns can in turn be representative of data issues or technical artefacts, e.g. batch effects, low quality events or outlier sample(s), ... or of biological signal of interest, given the scientific question at hand. Low dimensional data projection is also known as [DR](#), one of the main class of unsupervised [ML](#) algorithms (see Chapter 2).

Before introducing the most widely used low dimensional projection techniques for cytometry data, it is important to highlight some general considerations that drive the selection of an appropriate method.

First, when visualising multiple samples of a cytometry dataset, it is necessary to define the target level of representation. Two main approaches can be distinguished in this context (Figure 4.11):

- working at event level, i.e. representing single cells of one sample - or an aggregate of several samples, or
- working at sample level, i.e. projecting samples as single points, to assess distances or similarities between them.

In the first scenario, the primary objective is to identify distinct cell populations within a given sample - or aggregate of samples - and to evaluate the ‘proximity’ (see below) of these populations in the projected space. In contrast, the second scenario focuses on assessing proximity between entire set of samples in the study, with the aim of uncovering patterns that may relate to the experimental design, conditions of interest, or other relevant variables in metadata.

Second, it is necessary to attach a meaning to the notion of ‘proximity’ that the projection is intended to preserve. Some [DR](#) methods focus on preserving

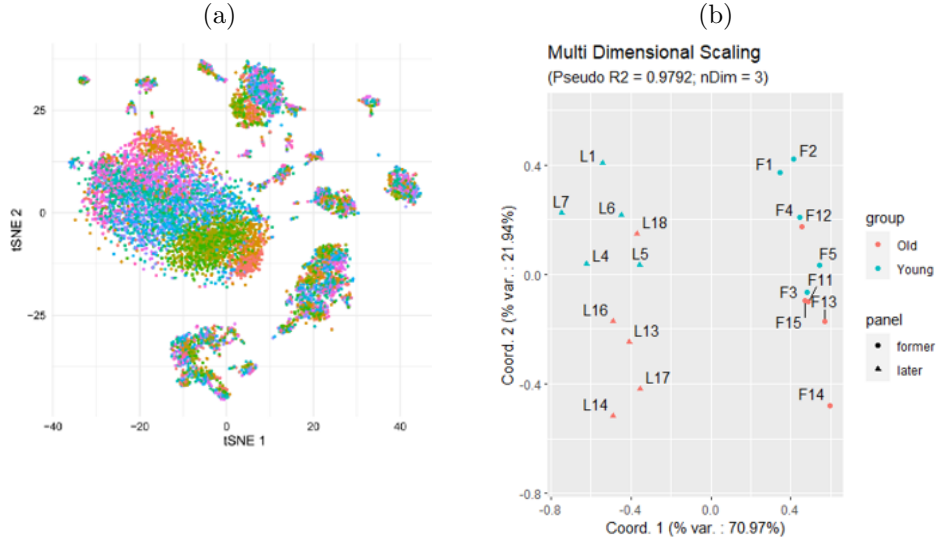


Figure 4.11: Two different approaches of low dimensional projection. (a) event level: i.e. representing single cells of one sample - or a sample aggregate. Here, dot of different colours correspond to different data acquisition batches. Figure reproduced from [90]; (b) sample level: i.e. projecting samples of a dataset as single points.

quantitative distances between data points - thus maintaining the global structure of the data - while others prioritise preserving the composition of each point close neighbour set, thereby emphasising more local structure preservation. From a methodological standpoint, preserving local structure is generally a less demanding task, and projection methods with this focus often require fewer dimensions to achieve a satisfactory representation of the high dimensional data. Conversely, preserving global distances can be more challenging and, in some cases, may not even be meaningful. Specifically, in very high dimensional spaces, the *curse of dimensionality* principle implies that pairwise distances tend to become increasingly similar [98], rendering absolute distance values uninformative for assessing similarity or dissimilarity. However, this issue is less pronounced for cytometry data, where the number of features (i.e. markers) typically remains within a few dozens.

Third, it is essential to assess the quality of the projection when applying a method. To this end, diagnostic tools are desirable to evaluate how well the projection maintains the intended structure. Such tools include quantitative metrics or graphical diagnostics that assess the quality of the low dimensional representation.

Fourth, an effective projection method should ideally offer tools for interpreting the resulting plots. This includes mechanisms for relating spatial features of the projection - such as point locations, axes, or directions - to underlying data characteristics, such as feature importance or associations with external variables.

4.4.2 Principal Component Analysis

Principal Component Analysis ([PCA](#)) [99] is the oldest and probably most famous [DR](#) method. It is a linear algorithm that project high dimensional data into a new orthogonal coordinate system. The new axes, or *principal components*, are linear combinations of the original axes such that they capture the maximum possible variance in the data. As such, for a given selected number of principal components (e.g. two), the [PCA](#) algorithm aims at preserving, as much as possible, the global structure (geometry) of the data, i.e. the angles and euclidean distances between data points². [PCA](#) is a linear algebra based deterministic method. As such, it always gives the same results when applied on the same inputs, which is an advantage over the non linear methods discussed in the next section.

Importantly, [PCA](#) requires the input data to be in the form of a rectangular matrix, where each observation is described by a fixed set of numeric features. With this requirement, [PCA](#) may be applied to visualise cytometry data at the event level, where individual cells are treated as observations. However, [PCA](#) is not suitable for projecting data at the sample level, since each cytometry sample itself is a data matrix - containing many events - rather than a single vector of numerical values.

4.4.3 tSNE and UMAP

t-distributed Stochastic Neighbour Embedding ([tSNE](#)) [100], and Uniform Manifold Approximation and Projection ([UMAP](#)) [101] are two widely used non linear [DR](#) methods that both aim at preserving the composition of point neighbourhoods. This means that points that are close in high dimensions remain, as much as possible, close in the lower-dimensional embedding (data

²More exactly, [PCA](#) preserves scalar products between data points, which are closely related to distances and angles.

representation), often revealing clusters and manifolds³.

[tSNE](#) and [UMAP](#) are widely used for generating low dimensional representations of cytometry data at the single-cell (event) level. As these methods are non linear and prioritise the preservation of local structure - which, as mentioned, is less demanding than distance preservation, they often produce more meaningful visualisations in two or three dimensions than linear methods like [PCA](#). However, it is important to note that in such embeddings, distances between distant points lack interpretability, and no conclusions should be drawn about global relationships in the data.

Additionally, using [tSNE](#) and [UMAP](#) is more complex than [PCA](#) due to the following factors:

- Both are iterative and stochastic algorithms, meaning that different random initialisations, i.e. different initial embeddings, can lead to different outcomes.
- Their behaviour is highly sensitive to one key parameter that controls the size of the neighbourhood used to preserve local structure: for [tSNE](#), this parameter is the *perplexity*, and for [UMAP](#), it is a ‘number of neighbours’ parameter.

As an illustrative example, Figure 4.12 shows the projection of the events of the same cytometry sample using [PCA](#), [tSNE](#) and [UMAP](#) [102], where the cells have been coloured according to immune cell populations (cell annotations were obtained by manual gating - see Section 4.5.1). On these 2D low dimensional projection plots, it is clear that both [tSNE](#) and [UMAP](#) tend to better highlight clusters corresponding to cell populations, by separating them more clearly than [PCA](#).

Finally, let us mention that there is a recurrent debate about which method, among [tSNE](#) and [UMAP](#), is ‘better’ for visualisation of single-cell data. Notably, a relatively widespread claim among practitioners is that [UMAP](#) could do a better job at preserving the global structure of the data, and is also faster to run than [tSNE](#) [103]. However, this was proven wrong, if both algorithms are initialised in a similarly informative way, and run with a recent implementation [104]. Besides, although the two methods were originally introduced, in their respective original article, from very different angles coming from different fields, it seems - in our personal opinion - that they are actually very

³Here, a manifold refers to a low dimensional, continuous and smooth surface embedded within a higher-dimensional space on which (parts of) the data approximately lie.

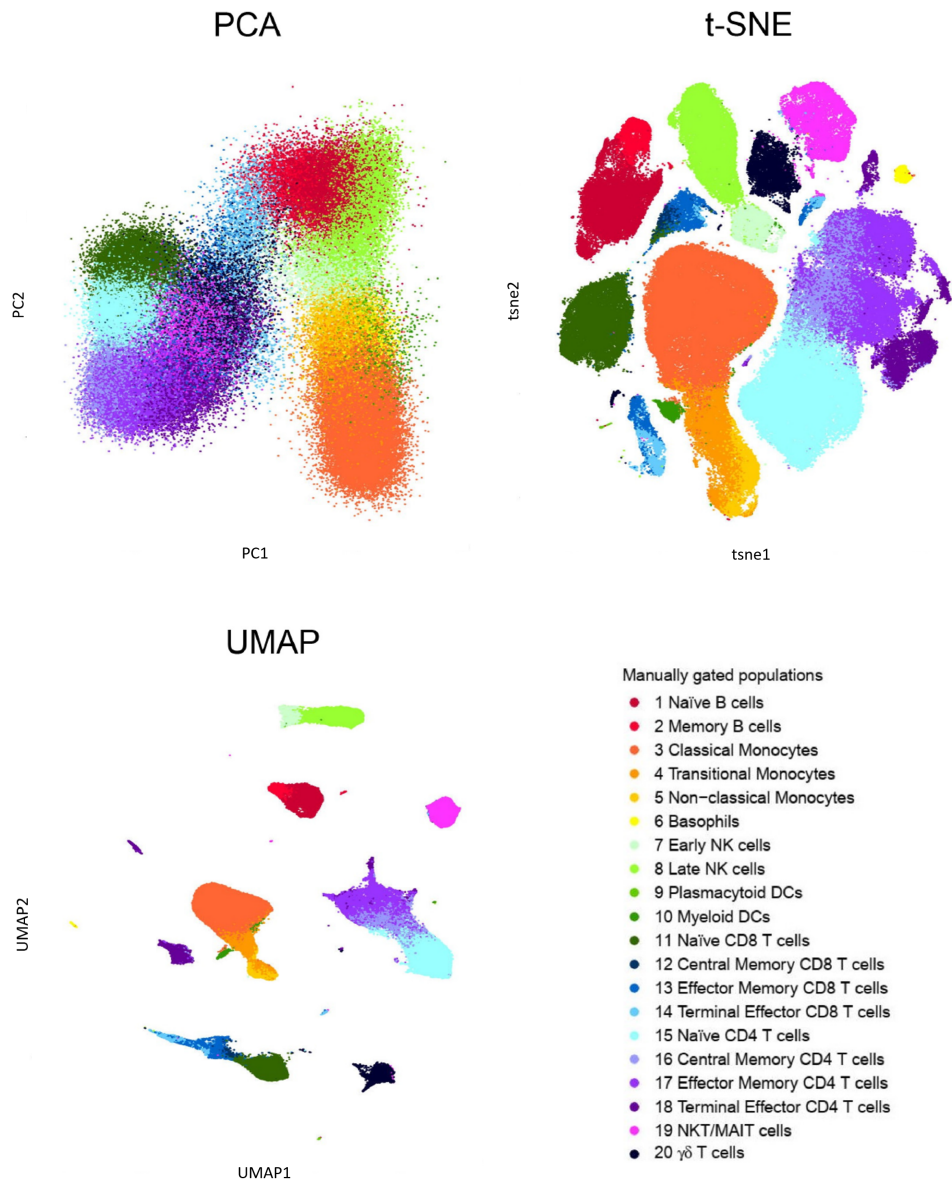


Figure 4.12: Visualisation, at event level, of the same flow cytometry sample using three different DR algorithms: PCA, tSNE and UMAP. The input data consists of 184,968 CD45+ cells, and 21 markers are used. Manually gated subpopulations are coloured across the three plots. Figure adapted from [102].

similar from a practical point of view and in fact - everything else being equal - tend to produce similar outcomes. The latter seems to be in line with recent research efforts, aiming at unifying the two methods into a common framework ([105], [106], [107]).

4.4.4 Metric Multi-Dimensional scaling

The term Multi-Dimensional Scaling (MDS) actually refers to a family of DR methods that aim at distance preservation [108]. These methods calculate low dimensional embeddings, starting from a pairwise distance matrix (in the original high dimensional space) between data points. The oldest method in the MDS family is the *Classical Metric MDS* [109], which can be applied when the pairwise distance matrix is *euclidean*, i.e. when the distances can be calculated, using the euclidean distance formula, from a single data point matrix. In that case, the obtained projection coincides with the one obtained using PCA.⁴

Metric MDS can be seen as a generalisation of Classical Metric MDS that can be applied when the input pairwise distance matrix is not necessarily euclidean, but still has the properties of a distance metric [108]. Metric MDS then minimises a criterion called *stress*, which represents a quadratic error of projection and is defined as follows:

$$\sigma(\mathbf{X}) = \sum_{i < j} (\delta_{ij} - d_{ij}(\mathbf{X}))^2 \quad (4.4)$$

where $\mathbf{X}$ is the low dimensional embedding, δ_{ij} is the (high dimensional) distance between sample i and sample j , and d_{ij} is the euclidean distance between the corresponding projections in $\mathbf{X}$.

The Metric MDS method is interesting for obtaining low dimensional projections of cytometry data **at sample level**. Indeed, since the input data is not a matrix, PCA - or Classical Metric MDS - cannot be applied in that context. However, since each cytometry data sample can be seen as a statistical sample of a multidimensional marker intensity distribution - with a number of dimensions equal to the number of markers, one can calculate pairwise distances between these distributions, and Metric MDS can then be applied. This idea is actually at the basis of the *CytoMDS* method, described in Chapter 8. In the following, we provide some insight about the concept of distance between cytometry samples.

⁴possibly up to one or several reflections.

Probability binning distances

Probably the most intuitive way to calculate a distance between two sample distributions consists in discretising the probability distributions into bins having the same boundaries, and comparing the obtained histograms, by aggregating the observed probability mass differences, bin by bin. Such an approach is illustrated in Figure 4.13, for the 1D case. The obtained distance is then referred to as a *probability binning distance* [110], of which numerous variants exist (chi square distance, Kullback-Leibler divergence...).

Generalising from the 1D to the multidimensional case is not trivial because bins must be defined jointly across all dimensions, resulting in an exponential increase in the total number of bins as dimensionality grows. A possible simplification is to approximate multidimensional distances by summing the corresponding 1D distances, but this comes at the cost of disregarding correlations between dimensions.

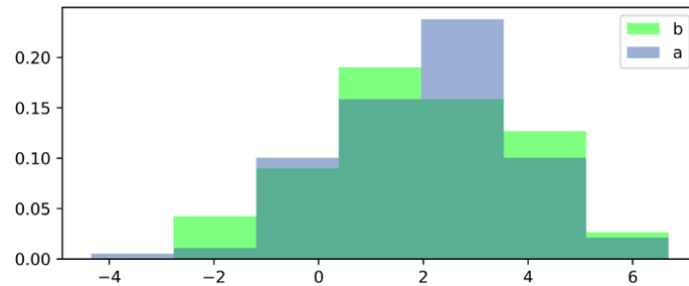


Figure 4.13: Calculating a distance between the distribution a (in blue) and the distribution b (in light green) using probability binning. The dark green area corresponds to common mass for each bin, so that the calculated distance is the sum of the leftover blue and light green areas.

Importantly, probability binning distances can lead to counter-intuitive results, when used to compare cytometry samples. Indeed, in cytometry, practitioners are mostly interested in identifying differences in terms of cell populations, which typically translates into presence or absence of a mode of the distribution, or into different relative probability mass of some parts of the intensity distribution. Less important to the practitioners are e.g. (small) shifts between sample distributions, which can be due to technical artefacts linked to changes in instrument set-up, subtle differences in staining efficiency or other types of batch or sample effects. Nevertheless, as is illustrated in Figure 4.14, probability binning can lead to higher distances between distributions

that differ only by a small shift, than, for instance, between distributions having a different number of modes.

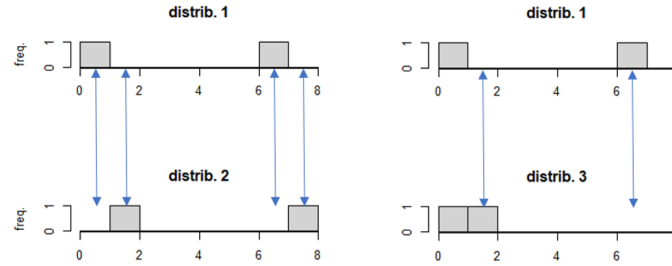


Figure 4.14: Calculation of a probability binning distance between simple distributions. On the left, distributions 1 and 2 are the same, up to a one unit shift. However, the probability binning distance between the two distributions is maximum, i.e. 4 units, as bins from exactly the same location are compared one-to-one, as indicated with the arrows. On the right, distributions 1 and 3 are conceptually very different, i.e. when considering the number of modes, but the probability binning distance between these two distributions is only two units, as highlighted by the arrows.

Earth Mover's Distance

The **EMD**, a.k.a. the order-1 Wasserstein distance, between two samples is defined as the minimal cost needed to transport the distribution mass of one sample to the other (Figure 4.15). In that definition, the ‘transportation cost’ of one unit of mass from location x to location y is itself computed as the Euclidean distance between x and y [111].

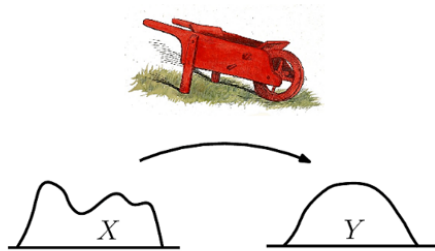


Figure 4.15: Principle of the **EMD**: it can be computed as the minimal cost (or effort) needed for transporting the distribution mass of one distribution to the other. Figure: credit to F. Cazals, T. Dreyfus and D. Mazauric (https://sbl.inria.fr/doc/group_Earth_mover_distance-package.html).

Figure 4.16 illustrates the better suitability of the **EMD** over probability

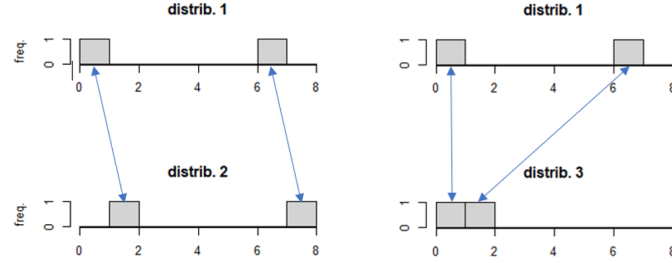


Figure 4.16: Calculation of the [EMD](#) between simple distributions. On the left, distributions 1 and 2 are the same, up to a one unit shift. The calculated [EMD](#) is only 2 units (=2 mass units moved by a 1 unit distance). On the right, the [EMD](#) between distributions 1 and 3, having different numbers of mode, is 5 units (=1 mass unit moved by a 5 unit distance).

binning distances for cytometry sample comparison, reusing the same example as in Figure 4.14. Note that other authors - e.g. [112] - have confirmed the suitability of [EMD](#) for comparing cell populations.

Therefore, when developping the *CytoMDS* method for visualisation of cytometry samples (Chapter 8), we chose [EMD](#) as a sample dissimilarity measure, and used Metric [MDS](#) for the low dimensional projection.

4.4.5 Assessing the quality of projection

As mentioned in Section 4.4.1, it is essential to check the quality of obtained low dimensional projections. This assessment can be done visually using a plot, and/or numerically using quality metrics. The choice of metric can be tailored as to focus to what is the most important for the user, e.g. local structure and/or global structure preservation.

Here we provide two examples of projection quality assessment tools, i.e. a generic, scale-independent quality criterion measuring neighbourhood preservation, and a plot that can be used specifically to assess the quality of distance preserving methods.

4.4.5.1 Scale independent quality criterion for neighbourhood preservation

Lee and Verleysen [113] proposed a generic quality plot that can be used for any [DR](#) method, and from which scale independent quality metrics can be de-

rived. The plot is based on assessing neighbourhood preservation for different values of the considered number of neighbours, i.e. for different scales.

Formally, given a set of N high dimensional data points: $i : i = 1, \dots, N$, and given an embedding X - of which the quality is to be evaluated - let us designate by:

- K , the number of neighbours in a considered neighbourhood (K varying from 1 to N);
- ν_i^K , the set of data points that are among the K nearest neighbours of data point i in the original (high dimensional) space;
- n_i^K , the set of data points that are among the K nearest neighbours of data point i in the low dimensional embedding.

Let us then assess the quality of the embedding X , for a given scale K by:

$$Q_{NX}(K) = \sum_{i=1}^N \frac{|\nu_i^K \cap n_i^K|}{KN} \quad (4.5)$$

$Q_{NX}(K)$ measures the average proportion of K size neighbours that get preserved in the low dimensional embedding, for all data points. We can then calculate the relative quality of the embedding, as compared to the average quality of a randomly chosen embedding by:

$$R_{NX}(K) = \frac{(N-1)Q_{NX}(K) - K}{N-1-K} \quad (4.6)$$

This latter quality indicator is usually comprised between 0 and 1⁵. Plotting it as a function of K , on a logarithmic scale, as in Figure 4.17, gives a good idea of the performance of the embedding as a function of number of neighbours (K).

⁵Technically, it can be negative if the observed quality of the embedding is less than the average quality of a randomly chosen embedding.

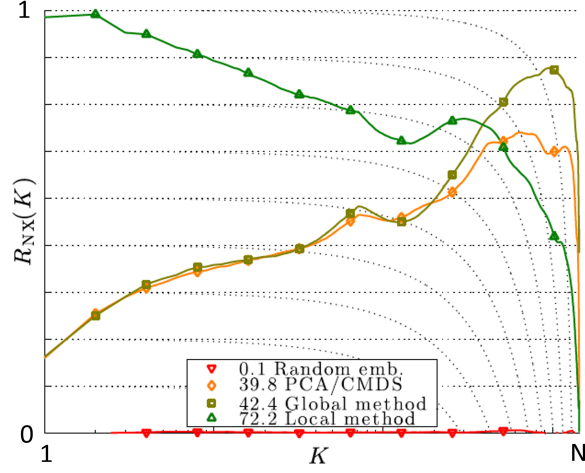


Figure 4.17: Example of curves measuring the relative quality of various embeddings, as a function of scale (K , varying from 1 to N). The red flat curve corresponds to a random embedding; the orange curve of diamonds corresponds to [PCA](#) which is equivalent to Classical Metric [MDS](#) (CMDs on the plot); the yellow-green curve of squares corresponds to a method preserving the global structure (for example metric [MDS](#)); the green curve of triangles corresponds to a method preserving local structure (typically [tSNE](#)). Displayed numbers in the legend show the corresponding [AUC](#). Horizontal dashed straight lines correspond to constant values of $R_{NX}(K)$ (Equation 4.6) while dashed curves correspond to constant values of $Q_{NX}(K)$ (Equation 4.5). Figure adapted from [114].

Then, using the values of that curve, global quality indicators, independent of scale K , can then be computed, e.g. the [AUC](#):

$$AUC = \frac{\sum_{K=1}^{N-2} R_{NX}(K)/K}{\sum_{K=1}^{N-2} 1/K} \quad (4.7)$$

4.4.5.2 Shepard diagram

The Shepard diagram is a plot that can be used to assess the projection quality of distance preserving methods, e.g. [MDS](#), and which resemble the residual plot familiar from regression analysis [115]. It is a scatter plot, with each dot representing a pair of data points. The x value is the high dimensional distance between the two data points and the y value is the obtained euclidean distance on the low dimensional projection. A projection of high quality results in most points being close to the ideal 45 degree identity line. An example is shown in Figure 4.18.

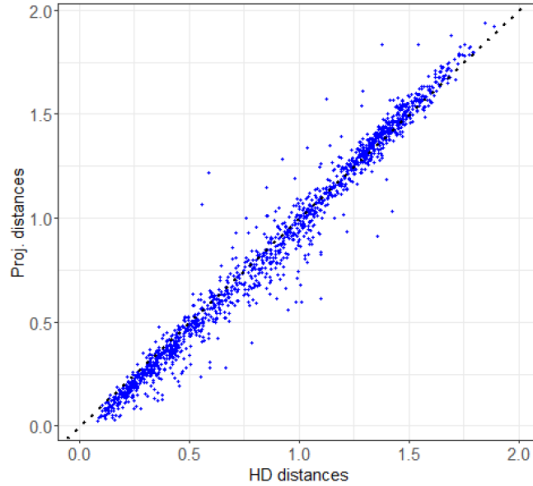


Figure 4.18: Example of Shepard diagram. Figure from [116].

4.5 Cell population identification

4.5.1 Overview

After preprocessing and QC, the next step (see Figure 4.1) is to assign the remaining events of each sample to cell populations. These cell populations are typically nodes of a hierarchical classification of immune cells, such as in Figure 4.19. Depending on the application and research question, the investigator will concentrate on particular nodes within the hierarchy - for example, to compare the frequencies of the relevant cell populations of interest across different samples.

In cytometry, the cell population identification process is usually called *gating*, which refers to the selection of successive subpopulations of cells for analysis [117]. Traditionally, gating has been carried out manually, relying on expert knowledge of cellular characteristics such as size, granularity, and the expression levels of various markers included in the cytometry panel. These marker expression patterns are annotations indicating marker positivity or negativity, associated with the different cell population nodes in the hierarchy (see Figure 4.19). Section 4.5.2 provides a detailed example of manual gating performed step by step.

Over the past decade, automated and semi-automated algorithms have been developed to replace or complement manual gating. These approaches are discussed in Section 4.5.3.

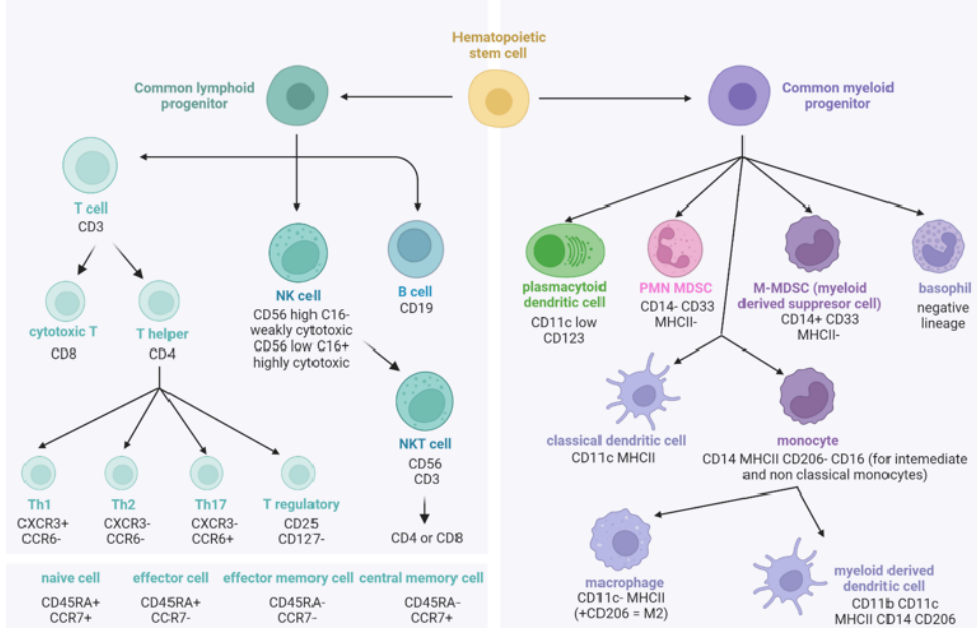


Figure 4.19: Hierarchy of hematopoietic immune cells, here focusing on PBMCs. Figure: credit to Elena Benidovskaya.

4.5.2 Manual gating example

Just like cell population classification, the gating procedure is hierarchical, and it is also iterative. Based on the initial sample set of events, the practitioner selects an ad-hoc 2D view - selection of suitable x and y axes - and manually sets one (or several) filter(s) to select the subpopulation(s) of interest based on *gates*, i.e. polygons drawn on the plot. This can be done using dedicated softwares (see Section 5.2). The user then selects one of the newly defined subpopulations, decides on a new 2D view to represent the selected subpopulation, and draws a new (set of) gate(s). This is repeated as many times as needed in order to define the (set of) subpopulation(s) of interest.

In the following, we illustrate the manual gating procedure by providing a practical example. The context is the immuno-phenotyping of human PBMC samples, focusing on rare Myeloid Derived Suppressor Cells (MDSC) subpopulations. MDSC are a heterogeneous population of immature myeloid cells that participate to the immuno-suppressive function [118]. They are scarce

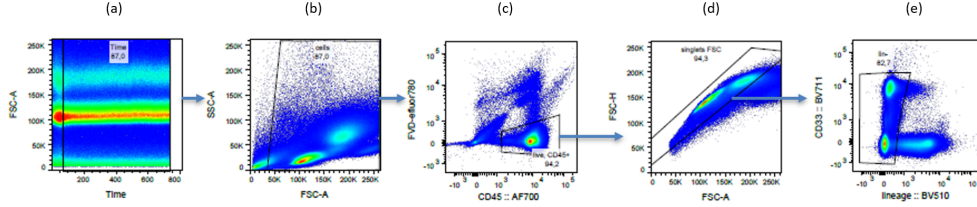


Figure 4.20: Manual gating example - pre-gating steps and selection of the cell population of interest.

under healthy condition but can accumulate under certain pathologies, e.g. some cancer types [119]. The gating strategy outlined below is described in detail in [120].

Figure 4.20 shows the first steps of the manual gating, which are actually some of the preprocessing steps described earlier in this chapter.

First (a), the FSC-A signal - as a function of time - is examined, and a rectangular gate is drawn to filter out the beginning of the data acquisition, where the signal is not yet stable, leading to keeping 87% of the events. This corresponds to ‘QC in time’ (see Section 4.3.4). Then (b), these events are displayed in a (FSC-A, SSC-A) 2D view, in order to gate out the debris (see Section 4.3.5.3), and define an event set named ‘cels’. Subsequently (c) the ‘cels’ population is represented in a (CD45, FVD) 2D view, which allows to create a ‘live, CD45+’ population. This is actually the dead cell removal step described in Section 4.3.5.4. Note that here, the user has chosen to define a gate of a specific form, to precisely extract what they think is the cluster of interest. Then (d), the ‘live, CD45+’ population is displayed in a (FSC-A, FSC-H) view, to manually remove doublets and multiplets - as explained in Section 4.3.5.2, defining a ‘singlets FSC’ population. And finally, yet another filtering gate is applied to the ‘singlets FSC’ population to keep only the cell populations under study. For this, the event intensities measured in a ‘lineage’ channel, which combines the expression of both markers CD19 and CD56, are used. Keeping the cells that have a low expression for these two markers effectively gates out both B cells and NK cells, keeping only T cells and myeloid-derived cells.

Following these *pre-gating* steps, i.e. gating operations related to preprocessing and QC, the gating procedure continues, as is shown in Figure 4.21. In (f), the cell population of interest is split into a ‘CD3+’ subpopulation (corresponding to T cells) and ‘CD33+’ sub- population (corresponding to

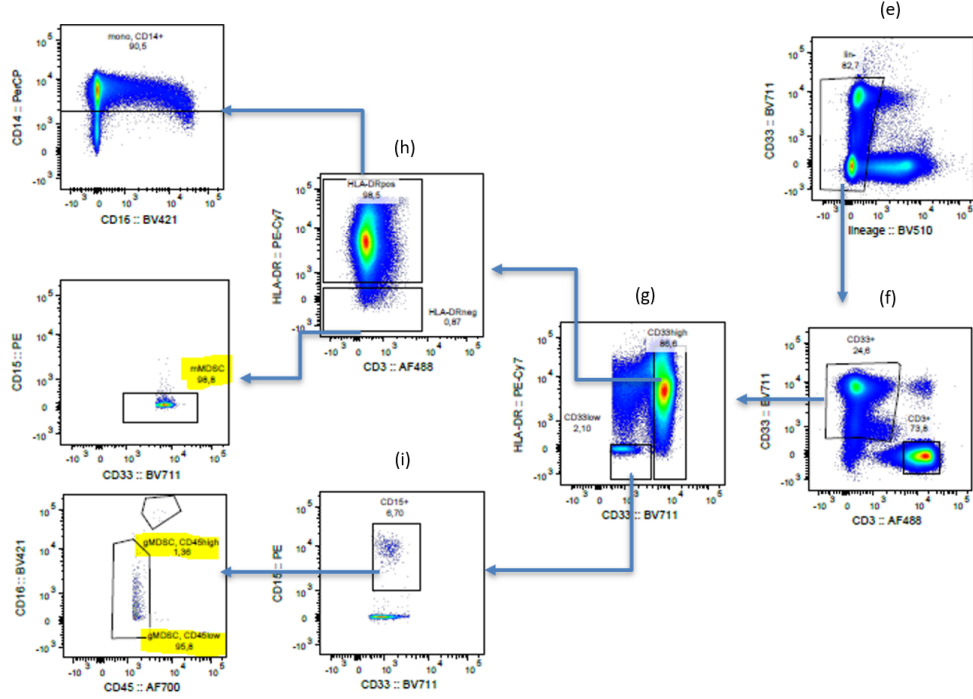


Figure 4.21: Manual gating example - delving into refined subsets of MDSC cells. Gating strategy from [120].

myeloid-derived cells). In (g), the ‘CD33+’ is split into 2 other branches: the ‘CD33high’, and the ‘CD33low’ populations, and these, in turn, are further gated and split in further subpopulations - (h) and (i). Finally, various MDSC populations of interest are obtained (highlighted in yellow in Figure 4.21).

The numerous steps described above highlight the complexity of the manual gating procedure, and the requirement for advanced biological expertise to determine the appropriate sequence of marker intensity combinations that allow accurate identification of the target cell populations relevant to the study. This complexity justifies the use of the term ‘gating strategy’. Moreover, several steps depicted in Figures 4.20 and 4.21 involve the application of polygonal gates - sometimes with intricate shapes - raising the question of whether the exact placement and contours of these gates may sometimes be somewhat subjective or arbitrary.

4.5.3 Automated and semi-automated approaches

These methods can be split into 3 categories:

- clustering methods (Section 4.5.3.1)
- (supervised) automated gating methods (Section 4.5.3.2)
- semi-supervised methods (Section 4.5.3.3)

4.5.3.1 Clustering

Clustering methods are unsupervised algorithms that aim at grouping events into cell populations without using any prior biological knowledge. For biological interpretation of the obtained clusters, an additional annotation step - mostly manual - will need to be performed as a postprocessing.

There exist numerous clustering methods for cytometry data, and discussing them all is beyond the scope of this introductory chapter. As an illustration, we describe herebelow the main principles and steps of one of the most widely used methods: *FlowSOM* [121, 122].

The FlowSOM workflow consists of 4 consecutive steps (Figure 4.22):

1. running preprocessing and QC for each cytometry Flow Cytometry Standard (FCS) file (see Section 5.1.1);
2. performing the main clustering, which is based on a Self-Organising Map (SOM) algorithm [123]. The number of clusters is typically set higher than the expected number of cell populations, for fine-tuning of the interpretation and visualisation. Obtained cluster nodes can be interpreted visually, either by using star charts showing the mean marker values for each node - Figure 4.22 (a), or pie charts showing their concordance with a user-provided manual gating - Figure 4.22 (b);
3. connecting the obtained clusters into a Minimal Spanning Tree (MST) [124], resulting in a connected acyclic graph, further enhancing visualisation;
4. performing *meta-clustering*, i.e. grouping similar clusters into bigger meta-clusters, with as goal to constitute a good starting point for annotating cell populations. For this, an extension of hierarchical clustering, i.e. *consensus hierarchical clustering* [125] is used.

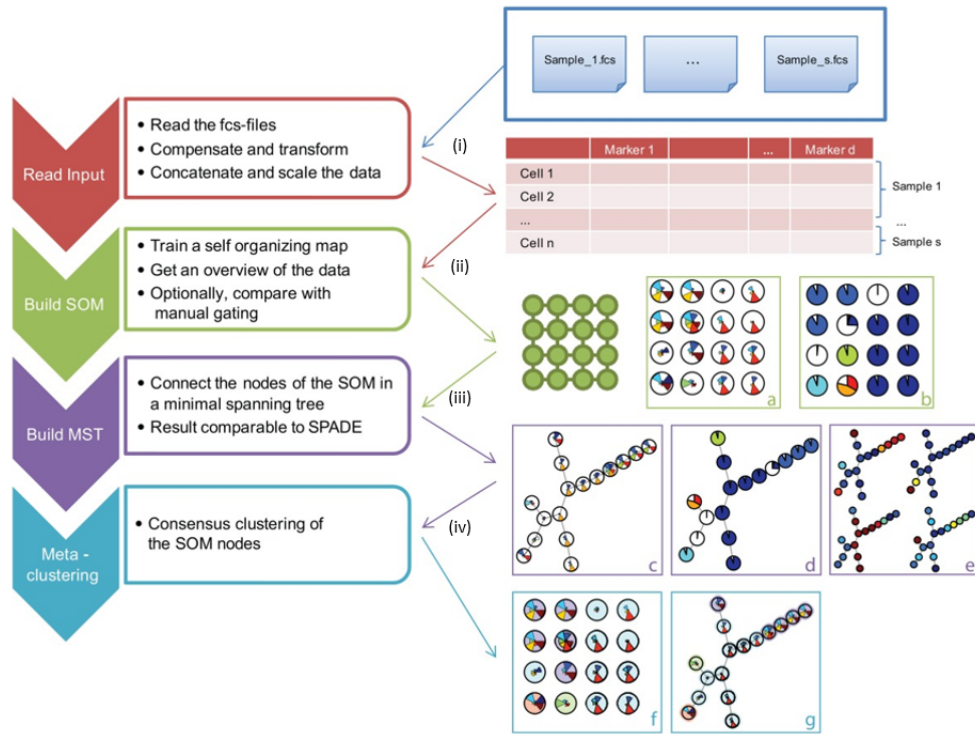


Figure 4.22: Overview of the four steps of the FlowSOM workflow. (i) The sample files are read and preprocessed, leading to a matrix where each row represents one cell, and describes the measured marker values in the different columns. (ii) A **SOM** is trained on the matrix, giving as output a grid of nodes, corresponding to cell clusters. Visualisations, consisting of (a) star charts showing the mean marker values of each node, and (b) pie charts showing the concordance with a manual gating, can be obtained. (iii) Cluster nodes are connected in a **MST**, which shows a better view of similarities between nodes. The star charts and pie charts can be added to the **MST** view as well (c, d, e). (iv) A consensus meta-clustering of the nodes is calculated, and indicated by additional background colours to the nodes, both in the grid view (f), or the **MST** view (g). Figure reproduced from [121]. SPADE: SPAnning tree progression of Density normalised Events, a cell population clustering method that was state of the art at the time of publication of the FlowSOM article [129].

According to reviews and benchmarkings [126, 102, 127, 128], the main advantages of *FlowSOM* are its short running time and its comprehensive visualisation tools, i.e. star charts, pie charts and possible overlay of meta-clusters, both in grid or **MST** views.

4.5.3.2 Automated gating

In (supervised) automated gating, the labels of previously manually gated samples - consisting of cell assignments to cell populations - are provided as input for training a ML model (see Section 2.3). This model can then be used to automatically gate other samples having the same set of markers, for classifying their cells in the same (sub)populations as in the training set.

Methods for automated gating include DL methods [130, 131], and more classical ML methods like decision trees [132].

4.5.3.3 Semi-supervised methods

Semi-supervised methods share similarities with clustering approaches but incorporate limited prior biological knowledge to guide the discovery of cell populations. Unlike supervised gating, this prior information does not include precise gate boundaries. Instead, it consists of loosely defined expectations - typically, the presence of high-level populations within a cellular hierarchy and their associated lineage markers (e.g. CD3+, CD4+, CD8- for helper T cells, or CD19+ for B cells).

Methods such as *Automated Cell type Discovery and Classification through knowledge transfer (ACDC)* [133] and the Bayesian tree-based approach described by Ji et al. [134] use this information in the form of marker profiles, i.e. structured tables indicating expected marker expressions for known cell types, such as in Figure 4.23.

Another semi-supervised algorithm is *Scyan* [135], which employs a Bayesian ANN model known as *normalising flow*. This method simultaneously classifies cells across multiple samples, corrects for batch effects, and supports the discovery of novel cell populations.

Compared to purely unsupervised methods, the semi-supervised approaches simplify the biological interpretation of the resulting clusters. By incorporating prior knowledge, they enable (partial) automated annotation of known populations, reducing the manual effort required to interpret clusters and focusing attention on potentially novel subpopulations.

	CD45	CD45RA	CD19	CD11b	CD4	CD8	CD34	CD20	CD33	CD38	CD3
Mature CD4+ T	+	-	-		+	-			-	-	+
Mature CD8+ T	+	-	-		-	+			-	-	+
Naive CD4+ T	+	+	-		+	-			-	-	+
Naive CD8+ T	+	+	-		-	+			-	-	+
CD11bhi Monocyte	+		-	+	-	-	-		+		-
CD11b- Monocyte	+		-	-	-	-	-		+		-
Pre-B I			+		-	-	+	-	-	+	-
Pre-B II			+		-	-	-	-	-	+	-
Immature B			+		-	-	-	-	-	-	-
Mature CD38lo B			+		-	-	-	+	-	-	-
Mature CD38mid B			+		-	-	-	+	-	+	-
NK	+	+	-		-	-			-	+	-
Plasma cell			+		-	-			-	+	-

Figure 4.23: Example of marker profile input table used in semi-supervised population identification methods.

4.6 Statistical analysis

4.6.1 Overview

Following the identification of cell populations, the last step of cytometry data analysis typically involves downstream statistical analysis (Figure 4.1). In this context, we use the term ‘statistical analysis’ broadly to refer to the application of classical statistical methods or ML techniques aimed at addressing the underlying scientific question.

Statistical analysis may include one or more of the following tasks:

- Computing descriptive statistics, such as per-sample cell population abundances, ratios between cell populations, or median marker expression levels within specific populations. Visualisation tools - such as heatmaps - can aid in the interpretation of these results [136].
- Comparing experimental conditions to identify statistically significant differences in cell population abundance - *Differential Abundance (DA)* - or in marker expression - *Differential Expression (DE)* [137]. The latter is also referred to as *Differential State (DS)*, as typically the markers being compared are state or functional markers (see Section 1.2), in contrast to lineage markers, which are used primarily for identifying cell types. Methods for these comparisons range from classical statistical tests (e.g. t-tests, non-parametric tests) to more advanced approaches [138, 139].

- Developing predictive models for outcomes of interest using [ML](#) algorithms (see Chapter 2).
- Constructing cell development models, which aim at inferring developmental trajectories from high dimensional cytometry data using unsupervised *seriation* techniques. These methods operate under the assumption that the dataset represents a snapshot of cells at various stages along a continuous developmental process, and seek to reconstruct the underlying trajectory based on gradients in the data [140].

4.6.2 Biomarker Discovery

In cytometry data analysis, *biomarker discovery* refers to the identification of cellular features - such as the abundance of specific immune cell subsets or the expression patterns of phenotypic markers - that are statistically associated with a biological condition, clinical outcome, or therapeutic response. In clinical settings, such biomarkers can serve multiple roles, including diagnosis, prognosis, stratification of patient cohorts, monitoring treatment responses, assessing drug or vaccine safety and efficacy, and risk profiling [140].

Several methodological approaches can be employed for biomarker discovery in cytometry. One strategy involves clustering the data using a population identification method (see Section 4.5), followed by statistical comparison of cluster-level cell counts or marker expression intensities across conditions. This approach corresponds to [DA](#) or [DE](#) analysis, as described in Section 4.6.1.

An alternative strategy involves performing more exhaustive clustering (i.e. over-clustering), followed by a feature selection step. These selected features can then serve as input variables for a [ML](#) model to predict the relevant outcome, as in *FloReMi.1* [141].

A third approach involves the use of end-to-end [DL](#) models. These models bypass explicit cell population identification - and sometimes even some of the traditional preprocessing steps - by using raw cytometry data as input to an [ANN](#) that directly predicts the outcome of interest. An example of such an approach is presented in Chapter 10.

Chapter 5

Bioinformatics infrastructure and software

A bioinformatics infrastructure consists of tools, workflows, and best practices designed to simplify the data analysis process. It abstracts away the complexities of software and data handling, allowing users to concentrate on the scientific aims of their analysis. By shifting the focus from technical implementation to analytical interpretation, this approach supports more effective and insightful exploration of the data [142].

In this chapter, we discuss some key elements of the bioinformatics infrastructure needed to analyse cytometry data.

5.1 Cytometry data storage

5.1.1 Standard file format for cytometry data

The **FCS** file format [143] is the universally accepted standard for storing and exchanging data generated by flow cytometry instruments. Its primary goal is to ensure interoperability between flow cytometers, data analysis software, and researchers across different platforms and institutions. By establishing a consistent structure for recording marker intensity data, along with essential metadata, **FCS** allows for accurate, reproducible, and portable analysis of cytometric experiments.

The **FCS** format is built on a few core principles: self-description, flexibility, and compatibility. Each **FCS** file corresponds to one single cytometry sample, and is divided into 4 parts, called *segments* in the **FCS** terminology:

- A *header segment*, which is a text string that begins by identifying the version of the FCS standard used, followed by three pairs of byte offsets that designate the positions in the file of the next 3 segments.
- A *text segment* containing metadata (such as parameter names, acquisition settings, and compensation).
- A *data segment* containing the raw measurement values for each event (cell or particle).
- An optional *analysis segment* containing results or metadata generated during data analysis - typically by the acquisition or analysis software.

An important point is that the data segment - which typically contains the largest part of the file - is stored in binary format, making it not directly human-readable. As a consequence, while the text segment can be manually inspected, interpreting the full file typically requires specialised software (see below).

Notably, although the format was originally developed for fluorescence-based flow cytometry, it has been widely adopted for [CyTOF](#) as well, where the underlying data structure (events $\times$ parameters) remains compatible. The current version of the [FCS](#) format is 3.2.

5.1.2 Public dataset repositories

Public repositories for cytometry data provide centralised platforms for sharing, accessing, and reusing annotated flow and mass cytometry datasets, supporting transparency, reproducibility, and collaborative research for various disciplines using cytometry data. Here we restrict ourselves to briefly describing two widely used repositories, from which we took public datasets for use in this work.

FlowRepository [144] is a specialised public database designed for the storage and dissemination of flow and mass cytometry datasets, particularly those associated with peer-reviewed publications. It supports the upload of complete experiment metadata, [FCS](#) files, gating strategies, and analysis results. FlowRepository adheres to the *MIFlowCyt* standard (Minimum Information about a Flow Cytometry Experiment), ensuring that datasets are well-documented and interpretable. Researchers can search for datasets by author, publication, or experimental keywords, making it a valuable resource for method benchmarking, data re-analysis, and educational use.

Immunology database and analysis Portal (ImmPort) [145] is a broader data repository funded by the US National Institutes of Health (NIH) that includes a wide range of immunological data types, including clinical trial data, gene expression, and flow/mass cytometry datasets. It is part of an initiative to promote open access to data generated by NIH-funded research, especially in the context of human immunology. Cytometry data on ImmPort are often accompanied by rich metadata on experimental design, subject cohorts, and immunophenotyping panels, making it a powerful resource for integrated immunological analyses across multiple studies.

5.2 Turnkey software solutions for cytometry data analysis

Turnkey software solutions for cytometry data analysis are commercial, ready-to-use platforms designed to provide a user-friendly interface and a complete suite of tools for data visualisation, gating, and statistical analysis. Unlike open-source pipelines, which often require programming skills, manual configuration, and integration of multiple tools, turnkey software emphasises ease of use and accessibility.

Among the commercial tools available, *FlowJo* was occasionally used in our projects to facilitate collaboration with biologists. FlowJo, first released in the late 1990s, is one of the most widely used commercial platforms for analysing flow and mass cytometry data. Designed around a GUI, FlowJo enables users to import FCS files, perform manual gating, and visualise data through various plots (such as dot plots, histograms, and density plots). It supports batch processing, compensation adjustment, and statistical comparisons across samples or experimental groups, and is especially effective in generating analysis reports.

Gating strategies and analysis workflows in FlowJo are saved in a proprietary workspace file (.wsp), which records the gating hierarchy, applied transformations, compensation matrices, and display preferences. While the underlying FCS files remain unchanged, the workspace file acts as a project-level container that enables users to revisit, modify, or share their analysis pipelines with full traceability of the manual gating decisions. Notably, these .wsp files can also be parsed outside of FlowJo and used as inputs to open-source tools, enabling reproduction of the gating strategy within bioinformatics pipelines (Section 5.3.3).

5.3 Open science and reproducible research with Bioconductor

5.3.1 Open science and reproducible research principles

Open science and reproducible research principles are foundational to modern scientific inquiry. Therefore, striving to apply these best practices in our work represents a fundamental pillar of our project.

Open science can be defined as “the practice of science in such a way that others can collaborate and contribute, where research data, lab notes and other research processes are freely available, under terms that enable reuse, redistribution and reproduction of the research and its underlying data and methods” [146]. This openness fosters transparency, encourages collaborative development, and allows other researchers to critically evaluate, reuse, or build upon existing work. It helps to prevent redundant efforts and accelerates scientific progress by making tools and insights readily available.

Reproducibility refers to the ability of an independent researcher to obtain the same results using the original data and analysis procedures. In computational research, this means repeating an analysis using the same code and data, but performed independently by a different analyst [147, 148]. Achieving reproducibility depends not only on access to data and code, but also on clear and comprehensive documentation, version-controlled code, and standardised formats. When effectively implemented, these principles improve scientific integrity, promote reuse, and support long-term sustainability of research outputs.

5.3.2 Bioconductor project

The Bioconductor project [149, 150] is a cornerstone initiative in open and reproducible bioinformatics research. Launched in 2001, Bioconductor is an open-source software project based on the R programming language [151], designed to facilitate the analysis and comprehension of high-throughput biological data. It provides a curated ecosystem of interoperable packages developed by and for the scientific community, with applications spanning transcriptomics, epigenetics, proteomics, flow and mass cytometry, and more. The project exemplifies the principles of open science by ensuring that all code, documentation, and data are freely available to users.

Reproducibility is a core pillar of the Bioconductor ecosystem. Each pack-

age is required to follow strict guidelines for documentation, testing, and integration with the broader infrastructure. Packages must include reproducible examples, unit tests, and vignettes - extended documents that demonstrate how to use the software with real data. Furthermore, every package is built and tested nightly across multiple operating systems using standardised build systems. This ensures that software behaves consistently and robustly over time and across environments, and that any failures are quickly detected and addressed.

The Bioconductor submission process also acts as a quality filter. Packages undergo a formal review by experienced members of the community who assess not only functionality and performance, but also code clarity, documentation quality, and adherence to best practices. This peer review process encourages good software engineering and helps ensure that Bioconductor packages are reliable and maintainable. As a result, a package released through Bioconductor provides users with a high degree of confidence in its reliability, reproducibility, and alignment with the principles of open science. For developers, inclusion in the Bioconductor ecosystem represents both a mark of quality and an opportunity for broader collaboration and dissemination.

5.3.3 Bioconductor base packages for managing cytometry data

A major advantage of using Bioconductor is its provision of standardised infrastructure for core data representation and basic processing across various types of omics data. In the context of cytometry, foundational support is provided by packages such as *flowCore* [152] and *flowWorkspace* [153]. These packages define key object classes for representing cytometry data, including the *flowFrame* class for individual samples and the *flowSet* class for collections of samples. They also offer essential functionality for handling common pre-processing steps such as reading FCS files, performing compensation, applying scale transformations, and defining and applying manual gating strategies.

Another valuable Bioconductor package for cytometry workflows is *CytoML* [154], which enables the import of gating strategies from external platforms such as FlowJo (see Section 5.2). This package played a key role in the development of our pipelines, as it allowed manual gating procedures to be faithfully reproduced and seamlessly incorporated into otherwise automated data analysis workflows (see Chapter 7).

Chapter 6

Purpose of the research

In Chapter 4, we outlined the key steps involved in a cytometry data analysis pipeline, with particular emphasis on the distinction between manual analysis and automated algorithms. Despite the advent of high dimensional, high content cytometry, manual gating remains widely used. Beyond historical practices and resistance to change, this persistence may partly stem from the fact that manual gating allows researchers to directly identify cell populations of interest, guided by prior biological knowledge [126]. Moreover, because manual gates can be adjusted on a per-sample basis, this approach tends to be more resilient to batch effects compared to automated methods such as clustering.

Nevertheless, particularly in the context of high dimensional cytometry, manual gating suffers from several significant shortcomings:

- First, as the number of markers increases, exhaustive data exploration through manual gating becomes infeasible. This is due to the exponential growth in the number of possible 2D marker combinations [155]. For instance, Figure 6.1 illustrates this challenge for a 25-color panel, which yields 325 possible 2D plots - an unmanageable number for manual inspection. Moreover, this does not account for potentially meaningful patterns involving combinations of more than two markers, which is difficult to capture with manual gating.
- Second, the results of manual gating are often difficult to reproduce. Numerous multicenter studies have demonstrated that manual gating is one of the major sources of variability in flow cytometry experiments [8, 156]. This lack of reproducibility arises from both the complexity of the hierarchical data structure, and the inherent subjectivity of human in-



Figure 6.1: Schematic representation of all possible 2D views in a 25 colour panel.
Figure: credit to Simon Delandre.

terpretation. Indeed, subjective decisions come into play when selecting the sequence of 2D plots in the gating strategy and when defining gate boundaries [140].

- Third, manual gating tends to introduce bias toward expected outcomes, as it relies heavily on the researcher’s prior knowledge and assumptions. Consequently, biologically relevant patterns in the data may be missed or disregarded [155].
- Fourth, in large-scale experiments involving hundreds of samples, manual gating becomes especially time-consuming and labour-intensive.

Altogether, these limitations highlight the clear need for automated, reproducible, and data-driven analysis methods to replace or complement manual gating. Over the past 20 years, a wide range of automated algorithms have been developed to address the various steps of cytometry data analysis pipelines [140, 157] - see Chapter 4.

However, just like manual gating, automated data analysis presents its own set of challenges. These range from technical issues - such as parameter instability, sensitivity to data quality, and limited robustness across datasets - to broader conceptual concerns. For instance, constructing fit-for-purpose analysis pipelines that involve multiple interdependent steps can be difficult, ensuring the interpretability of complex model outputs can be challenging, etc. The first objective of this work is therefore to explore the different steps of a cytometry data analysis pipeline, with the aim of identifying common obstacles and pitfalls encountered when applying automated, data-driven methods. Particular attention is given to risks that may emerge when replacing manual expertise with algorithms.

In confronting these challenges, opportunities also arise to develop better tools and strategies. The second objective of this thesis is thus to design and implement novel computational methods tailored to some of the needs identified in the exploratory phase. In doing so, this work seeks to deliver concrete and practical outputs that can support real-world applications. Part II of this thesis - from Chapter 7 to Chapter 10 - provides a contextualised description of these methods and tools, along with selected use cases. The related challenges and risks that these approaches aim to address are examined in greater detail later, in Chapter 11.

Finally, a third and closely related objective is to promote transparency and reproducibility in cytometry data analysis. As a result, most methods and

tools developed in the course of this work are released as open-source software and integrated within the Bioconductor project (see Chapter [5](#)), aligning with the broader goal of contributing to sustainable and collaborative research.

References for the Introduction part

- [1] Katherine M McKinnon. “Flow Cytometry: An Overview”. en. In: *Curr. Protoc. Immunol.* 120 (Feb. 2018), pp. 5.1.1–5.1.11. DOI: [10.1002/cpim.40](https://doi.org/10.1002/cpim.40).
- [2] Leonard A Herzenberg et al. “The history and future of the fluorescence activated cell sorter and flow cytometry: a view from Stanford”. en. In: *Clin. Chem.* 48.10 (Oct. 2002), pp. 1819–1827. DOI: [10.1093/clinchem/48.10.1819](https://doi.org/10.1093/clinchem/48.10.1819).
- [3] J Paul Robinson. “Flow cytometry: past and future”. en. In: *Biotechniques* 72.4 (Apr. 2022), pp. 159–169. DOI: [10.2144/btn-2022-0005](https://doi.org/10.2144/btn-2022-0005).
- [4] Howard M Shapiro. *Practical Flow Cytometry*. en. 4th ed. Nashville, TN: John Wiley & Sons, June 2003.
- [5] Kieran O’Neill et al. “Flow cytometry bioinformatics”. en. In: *PLoS Comput. Biol.* 9.12 (Dec. 2013), e1003365. DOI: [10.1371/journal.pcbi.1003365](https://doi.org/10.1371/journal.pcbi.1003365).
- [6] Stephen P Perfetto, Pratip K Chattopadhyay, and Mario Roederer. “Seventeen-colour flow cytometry: unravelling the immune system”. en. In: *Nat. Rev. Immunol.* 4.8 (Aug. 2004), pp. 648–655. DOI: [10.1038/nri1416](https://doi.org/10.1038/nri1416).
- [7] Heddy Zola and Bernadette Swart. “CD Markers”. In: *Encyclopedia of Immunotoxicology*. Ed. by Hans-Werner Vohr. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1–5. ISBN: 978-3-642-27786-3. DOI: [10.1007/978-3-642-27786-3_216-2](https://doi.org/10.1007/978-3-642-27786-3_216-2).
- [8] Holden T Maecker, J Philip McCoy, and Robert Nussenblatt. “Standardizing immunophenotyping for the Human Immunology Project”. en. In: *Nat. Rev. Immunol.* 12.3 (Feb. 2012), pp. 191–200. DOI: [10.1002/cyto.b.20485](https://doi.org/10.1002/cyto.b.20485).

- [9] Pranab Dey. “Sample Preparation and Data Acquisition in Flow Cytometry”. In: *Diagnostic Flow Cytometry in Cytology*. Singapore: Springer Singapore, 2021, pp. 23–34. ISBN: 978-981-16-2655-5. DOI: [10.1007/978-981-16-2655-5_3](https://doi.org/10.1007/978-981-16-2655-5_3).
- [10] Andrew Reichard and Kewal Asosingh. “Best practices for preparing a single cell suspension from solid tissues for flow cytometry: Single cell suspension preparation”. en. In: *Cytometry A* 95.2 (Feb. 2019), pp. 219–226. DOI: [10.1002/cyto.a.23690](https://doi.org/10.1002/cyto.a.23690).
- [11] Andrea Cossarizza et al. “Guidelines for the use of flow cytometry and cell sorting in immunological studies (third edition)”. en. In: *Eur. J. Immunol.* 51.12 (Dec. 2021), pp. 2708–3145. DOI: [10.1002/cyto.a.23690](https://doi.org/10.1002/cyto.a.23690).
- [12] Mario Roederer. “Compensation in flow cytometry”. en. In: *Curr. Protoc. Cytom.* Chapter 1.1 (Dec. 2002), Unit 1.14. DOI: [10.1002/0471142956.cy0114s22](https://doi.org/10.1002/0471142956.cy0114s22).
- [13] Zofia Maciorowski, Pratip K Chattopadhyay, and Paresh Jain. “Basic multicolor flow cytometry”. en. In: *Curr. Protoc. Immunol.* 117.1 (Apr. 2017), pp. 5.4.1–5.4.38. DOI: [10.1002/cpim.26](https://doi.org/10.1002/cpim.26).
- [14] Mario Roederer and Attila Tárnok. “OMIPs—Orchestrating multiplexity in polychromatic science”. en. In: *Cytometry A* 77.9 (Sept. 2010), pp. 811–812. DOI: [10.1002/cyto.a.20959](https://doi.org/10.1002/cyto.a.20959).
- [15] Radwa Ali Mehanna. “Physical versus immunological purification of mesenchymal stem cells”. en. In: *Mesenchymal Stem Cells - Isolation, Characterization and Applications*. InTech, Nov. 2017. DOI: [10.5772/intechopen.69295](https://doi.org/10.5772/intechopen.69295).
- [16] Shafiuddin Siddiqui and Ferenc Livák. “Principles of advanced flow cytometry: A practical guide”. en. In: *Methods Mol. Biol.* 2580 (2023), pp. 89–114. DOI: [10.1007/978-1-0716-2740-2_5](https://doi.org/10.1007/978-1-0716-2740-2_5).
- [17] J Paul Robinson et al. “Flow cytometry: The next revolution”. en. In: *Cells* 12.14 (July 2023), p. 1875. DOI: [10.3390/cells12141875](https://doi.org/10.3390/cells12141875).
- [18] J P Robinson et al. “Multispectral Flow Cytometry: Next Generation Tools for Automated Classification”. en. In: *Microanal* 11.S02 (Aug. 2005), pp. 2–3. DOI: [10.1017/S1431927605510328](https://doi.org/10.1017/S1431927605510328).
- [19] Dmitry R Bandura et al. “Mass cytometry: technique for real time single cell multitarget immunoassay based on inductively coupled plasma time-of-flight mass spectrometry”. en. In: *Anal. Chem.* 81.16 (Aug. 2009), pp. 6813–6822. DOI: [10.1021/ac901049w](https://doi.org/10.1021/ac901049w).

- [20] Akshay Iyer, Anouk A J Hamers, and Asha B Pillai. “CyTOF® for the masses”. en. In: *Front. Immunol.* 13 (Apr. 2022), p. 815828. DOI: [10.3389/fimmu.2022.815828](https://doi.org/10.3389/fimmu.2022.815828).
- [21] Stéphane Chevrier et al. “Compensation of signal spillover in suspension and imaging mass cytometry”. en. In: *Cell Syst.* 6.5 (May 2018), 612–620.e5. DOI: [10.1016/j.cels.2018.02.010](https://doi.org/10.1016/j.cels.2018.02.010).
- [22] Evan W Newell and Yang Cheng. “Mass cytometry: blessed with the curse of dimensionality”. en. In: *Nat. Immunol.* 17.8 (July 2016), pp. 890–895. DOI: [10.1038/ni.3485](https://doi.org/10.1038/ni.3485).
- [23] Batta Mahesh. “Machine learning algorithms - A review”. In: *Int. J. Sci. Res. (Raipur)* 9.1 (Jan. 2020), pp. 381–386. DOI: [10.21275/art20203995](https://doi.org/10.21275/art20203995).
- [24] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th ed. Prentice Hall, 2020.
- [25] Gareth James et al. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013.
- [26] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Ed. by M Jordan, J Kleinberg, and B Schölkopf. Vol. 4. Information science and statistics 4. Springer, 2006. Chap. Graphical, p. 738. ISBN: 9780387310732.
- [27] Mikhail Belkin et al. “Reconciling modern machine-learning practice and the classical bias-variance trade-off”. en. In: *Proc. Natl. Acad. Sci. U. S. A.* 116.32 (Aug. 2019), pp. 15849–15854. DOI: [10.1073/pnas.1903070116](https://doi.org/10.1073/pnas.1903070116).
- [28] Peter I Frazier. “A tutorial on Bayesian optimization”. In: *arXiv [stat.ML]* (July 2018). URL: <http://arxiv.org/abs/1807.02811>.
- [29] Rita Lasfar and Gergely Tóth. “The difference of model robustness assessment using cross-validation and bootstrap methods”. en. In: *J. Chemom.* 38.6 (June 2024), e3530. DOI: [10.1002/cem.3530](https://doi.org/10.1002/cem.3530).
- [30] Pieter Kubben, Michel Dumontier, and Andre Dekker, eds. *Fundamentals of clinical data science*. en. 1st ed. Cham, Switzerland: Springer International Publishing, Jan. 2019. DOI: [10.1007/978-3-319-99713-1](https://doi.org/10.1007/978-3-319-99713-1).
- [31] Andrea Apicella, Francesco Isgrò, and Roberto Prevete. “Don’t push the button! Exploring data leakage risks in machine learning and transfer learning”. In: *arXiv [cs.LG]* (2024). DOI: [10.2139/ssrn.4733889](https://doi.org/10.2139/ssrn.4733889).

- [32] Francis Sahngun Nahm. “Receiver operating characteristic curve: overview and practical use for clinicians”. en. In: *Korean J. Anesthesiol.* 75.1 (Feb. 2022), pp. 25–36. DOI: [10.4097/kja.21209](https://doi.org/10.4097/kja.21209).
- [33] M M Rodríguez-Hernández, R E Pruneda, and J M Rodríguez-Díaz. “Statistical Analysis of the evolutive effects of language development in the resolution of mathematical problems in primary school education”. en. In: *Mathematics* 9.10 (May 2021), p. 1081. DOI: [10.3390/math9101081](https://doi.org/10.3390/math9101081).
- [34] David J Hand and Robert J Till. “A simple generalisation of the area under the ROC curve for multiple class classification problems”. en. In: *Mach. Learn.* 45.2 (Nov. 2001), pp. 171–186. DOI: [10.1023/a:1010920819831](https://doi.org/10.1023/a:1010920819831).
- [35] L I Lin. “A concordance correlation coefficient to evaluate reproducibility”. en. In: *Biometrics* 45.1 (Mar. 1989), pp. 255–268. DOI: [10.2307/2532051](https://doi.org/10.2307/2532051).
- [36] Patrick Schober, Christa Boer, and Lothar A Schwarte. “Correlation coefficients: Appropriate use and interpretation”. en. In: *Anesth. Analg.* 126.5 (May 2018), pp. 1763–1768. DOI: [10.1213/ANE.0000000000002864](https://doi.org/10.1213/ANE.0000000000002864).
- [37] F E Harrell Jr, K L Lee, and D B Mark. “Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors”. en. In: *Stat. Med.* 15.4 (Feb. 1996), pp. 361–387. DOI: [10.1002/\(SICI\)1097-0258\(19960229\)15:4<361::AID-SIM168>3.0.CO;2-4](https://doi.org/10.1002/(SICI)1097-0258(19960229)15:4<361::AID-SIM168>3.0.CO;2-4).
- [38] Abdallah Alabdallah et al. “The Concordance Index decomposition: A measure for a deeper understanding of survival prediction models”. en. In: *Artif. Intell. Med.* 148.102781 (Feb. 2024), p. 102781. DOI: [10.1016/j.artmed.2024.102781](https://doi.org/10.1016/j.artmed.2024.102781).
- [39] Enrico Longato, Martina Vettoretti, and Barbara Di Camillo. “A practical perspective on the concordance index for the evaluation and selection of prognostic time-to-event models”. en. In: *J. Biomed. Inform.* 108 (Aug. 2020), p. 103496. DOI: [10.1016/j.jbi.2020.103496](https://doi.org/10.1016/j.jbi.2020.103496).
- [40] Hajime Uno et al. “On the C-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data”. en. In: *Stat. Med.* 30.10 (May 2011), pp. 1105–1117. DOI: [10.1002/sim.4154](https://doi.org/10.1002/sim.4154).
- [41] E L Kaplan and Paul Meier. “Nonparametric estimation from incomplete observations”. en. In: *J. Am. Stat. Assoc.* 53.282 (June 1958), pp. 457–481. DOI: [10.1080/01621459.1958.10501452](https://doi.org/10.1080/01621459.1958.10501452).

- [42] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. en. In: *Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: [10.1007/bf02478259](https://doi.org/10.1007/bf02478259).
- [43] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033-295X. DOI: [10.1037/h0042519](https://doi.org/10.1037/h0042519).
- [44] Hong Hui Tan and King Hann Lim. “Vanishing gradient mitigation with deep learning neural network optimization”. en. In: *2019 7th International Conference on Smart Computing & Communications (ICSCC)*. IEEE, June 2019, pp. 1–4. DOI: [10.1109/icsc.2019.8843652](https://doi.org/10.1109/icsc.2019.8843652).
- [45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [46] Xinliang Zhu, Jiawen Yao, and Junzhou Huang. “Deep convolutional neural network for survival analysis with pathological images”. In: *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. Dec. 2016, pp. 544–547. DOI: [10.1109/BIBM.2016.7822579](https://doi.org/10.1109/BIBM.2016.7822579).
- [47] Jared L Katzman et al. “DeepSurv: personalized treatment recommender system using a Cox proportional hazards deep neural network”. en. In: *BMC Med. Res. Methodol.* 18.1 (Feb. 2018), p. 24. DOI: [10.1186/s12874-018-0482-1](https://doi.org/10.1186/s12874-018-0482-1).
- [48] Dong Wook Kim et al. “Deep learning-based survival prediction of oral cancer patients”. en. In: *Sci. Rep.* 9.1 (May 2019), p. 6994. DOI: [10.1038/s41598-019-43372-7](https://doi.org/10.1038/s41598-019-43372-7).
- [49] D R Cox. “Regression models and life-tables”. en. In: *J. R. Stat. Soc. Series B Stat. Methodol.* 34.2 (Jan. 1972), pp. 187–202. DOI: [10.1111/j.2517-6161.1972.tb00899.x](https://doi.org/10.1111/j.2517-6161.1972.tb00899.x).
- [50] Samar Abd ElHafeez et al. “Methods to analyze time-to-event data: The Cox regression analysis”. en. In: *Oxid. Med. Cell. Longev.* 2021.1 (Nov. 2021), p. 1302811. DOI: [10.1155/2021/1302811](https://doi.org/10.1155/2021/1302811).
- [51] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *ICLR Conference*. 2015. URL: <http://arxiv.org/abs/1412.6980>.
- [52] Y LeCun et al. “Backpropagation applied to handwritten zip code recognition”. en. In: *Neural Comput.* 1.4 (Dec. 1989), pp. 541–551. DOI: [10.1162/neco.1989.1.4.541](https://doi.org/10.1162/neco.1989.1.4.541).

- [53] Shengwen Zhou et al. “A hybrid framework for multivariate time series forecasting of daily urban water demand using attention-based convolutional neural network and long short-term memory network”. en. In: *Sustainability* 14.17 (Sept. 2022), p. 11086. DOI: [10.3390/su141711086](https://doi.org/10.3390/su141711086).
- [54] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *IEE proc.* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0143-7100.
- [55] Sean Meling Murray. “An exploratory analysis of multi-class uncertainty approximation in Bayesian convolution neural networks”. MA thesis. University of Bergen, 2018.
- [56] Farhana Sultana, A Sufian, and Paramartha Dutta. “Advancements in image classification using convolutional Neural Network”. In: *arXiv [cs.CV]* (May 2019). DOI: [10.1109/ICRCICN.2018.8718718](https://doi.org/10.1109/ICRCICN.2018.8718718).
- [57] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv [cs.LG]* (Feb. 2015). URL: <http://arxiv.org/abs/1502.03167>.
- [58] Azade Rezaeezade and Lejla Batina. “Regularizers to the rescue: fighting overfitting in deep learning-based side-channel analysis”. en. In: *J. Cryptogr. Eng.* 14.4 (Aug. 2024), pp. 609–629. DOI: [10.1007/s13389-024-00361-5](https://doi.org/10.1007/s13389-024-00361-5).
- [59] Mansheng Xiao et al. “Addressing overfitting problem in deep learning-based solutions for next generation data-driven networks”. en. In: *Wirel. Commun. Mob. Comput.* 2021.1 (Jan. 2021), pp. 1–10. DOI: [10.1155/2021/8493795](https://doi.org/10.1155/2021/8493795).
- [60] Maria Vakalopoulou et al. “Deep learning: Basics and convolutional neural networks (CNNs)”. en. In: *Machine Learning for Brain Disorders*. Neuromethods. New York, NY: Springer US, 2023, pp. 77–115. DOI: [10.1007/978-1-0716-3195-9_3](https://doi.org/10.1007/978-1-0716-3195-9_3).
- [61] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. en. In: *J. R. Stat. Soc. Series B Stat. Methodol.* 58.1 (Jan. 1996), pp. 267–288. DOI: [10.1111/j.2517-6161.1996.tb02080.x](https://doi.org/10.1111/j.2517-6161.1996.tb02080.x).
- [62] A.E. Hoerl. “Application of ridge analysis to regression problems”. In: *Chemical Engineering Progress* 58.3 (1962), pp. 54–59.
- [63] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *J. Mach. Learn. Res.* 15.56 (2014), pp. 1929–1958. DOI: [10.5555/2627435.2670313](https://doi.org/10.5555/2627435.2670313).

- [64] Connor Shorten and Taghi M Khoshgoftaar. “A survey on image data augmentation for deep learning”. en. In: *J. Big Data* 6.1 (Dec. 2019), pp. 1–48. DOI: [10.1186/s40537-019-0197-0](https://doi.org/10.1186/s40537-019-0197-0).
- [65] Thomas Liechti et al. “An updated guide for the perplexed: cytometry in the high-dimensional era”. en. In: *Nat. Immunol.* 22.10 (Oct. 2021), pp. 1190–1197. DOI: [10.1038/s41590-021-01006-z](https://doi.org/10.1038/s41590-021-01006-z).
- [66] Ana Margarida Araújo et al. “Metabolomic approaches in the discovery of potential urinary biomarkers of drug-induced liver injury (DILI)”. en. In: *Crit. Rev. Toxicol.* 47.8 (Sept. 2017), pp. 633–649. DOI: [10.1080/10408444.2017.1309638](https://doi.org/10.1080/10408444.2017.1309638).
- [67] C B Bagwell and E G Adams. “Fluorescence spectral overlap compensation for any number of flow cytometry parameters”. en. In: *Ann. N. Y. Acad. Sci.* 677 (Mar. 1993), pp. 167–184. DOI: [10.1111/j.1749-6632.1993.tb38775.x](https://doi.org/10.1111/j.1749-6632.1993.tb38775.x).
- [68] Fuqiang Li et al. “Autofluorescence contributes to false-positive intracellular Foxp3 staining in macrophages: a lesson learned from flow cytometry”. en. In: *J. Immunol. Methods* 386.1-2 (Dec. 2012), pp. 101–107. DOI: [10.1016/j.jim.2012.08.014](https://doi.org/10.1016/j.jim.2012.08.014).
- [69] Carlos P Roca et al. “AutoSpill is a principled framework that simplifies the analysis of multichromatic flow cytometry data”. en. In: *Nat. Commun.* 12.1 (May 2021), p. 2890. DOI: [10.1038/s41467-021-23126-8](https://doi.org/10.1038/s41467-021-23126-8).
- [70] David Novo. “A comparison of spectral unmixing to conventional compensation for the calculation of fluorochrome abundances from flow cytometric data”. en. In: *Cytometry A* 101.11 (Nov. 2022), pp. 885–891. DOI: [10.1002/cyto.a.24669](https://doi.org/10.1002/cyto.a.24669).
- [71] Bernd Bodenmiller et al. “Multiplexed mass cytometry profiling of cellular states perturbed by small-molecule regulators”. en. In: *Nat. Biotechnol.* 30.9 (Sept. 2012), pp. 858–867. DOI: [10.1038/nbt.2317](https://doi.org/10.1038/nbt.2317).
- [72] Eli R Zunder et al. “Palladium-based mass tag cell barcoding with a doublet-filtering scheme and single-cell deconvolution algorithm”. en. In: *Nat. Protoc.* 10.2 (Feb. 2015), pp. 316–333. DOI: [10.1038/nprot.2015.020](https://doi.org/10.1038/nprot.2015.020).
- [73] Helena L. Crowell et al. *CATALYST: Cytometry dATa anALYSis Tools*. R package version 1.32.0. 2025. DOI: [10.18129/B9.bioc.CATALYST](https://doi.org/10.18129/B9.bioc.CATALYST). URL: <https://bioconductor.org/packages/CATALYST>.
- [74] Holden T Maecker et al. “Selecting fluorochrome conjugates for maximum sensitivity”. en. In: *Cytometry A* 62.2 (Dec. 2004), pp. 169–173. DOI: [10.1002/cyto.a.20092](https://doi.org/10.1002/cyto.a.20092).

- [75] David R Parks, Mario Roederer, and Wayne A Moore. “A new “Logicle” display method avoids deceptive effects of logarithmic scaling for low signals and compensated data”. en. In: *Cytometry A* 69.6 (June 2006), pp. 541–551. DOI: [10.1002/cyto.a.20258](https://doi.org/10.1002/cyto.a.20258).
- [76] Wayne A Moore and David R Parks. “Update for the logicle data scale including operational code implementations”. en. In: *Cytometry A* 81.4 (Apr. 2012), pp. 273–277. DOI: [10.1002/cyto.a.22030](https://doi.org/10.1002/cyto.a.22030).
- [77] Surajit Ray and Saumyadipta Pyne. “A computational framework to emulate the human perspective in flow cytometric data analysis”. en. In: *PLoS One* 7.5 (May 2012), e35693. DOI: [10.1371/journal.pone.0035693](https://doi.org/10.1371/journal.pone.0035693).
- [78] Greg Finak et al. “Optimizing transformations for automated, high throughput analysis of flow cytometry data”. en. In: *BMC Bioinformatics* 11 (Nov. 2010), p. 546. DOI: [10.1186/1471-2105-11-546](https://doi.org/10.1186/1471-2105-11-546).
- [79] Ariful Azad, Bartek Rajwa, and Alex Pothen. “flowVS: channel-specific variance stabilization in flow cytometry”. en. In: *BMC Bioinformatics* 17 (July 2016), p. 291. DOI: [10.1186/s12859-016-1083-9](https://doi.org/10.1186/s12859-016-1083-9).
- [80] Rita Folcarelli et al. “Transformation of multicolour flow cytometry data with OTflow prevents misleading multivariate analysis results and incorrect immunological conclusions”. en. In: *Cytometry A* 101.1 (Jan. 2022), pp. 72–85. DOI: [10.1002/cyto.a.24491](https://doi.org/10.1002/cyto.a.24491).
- [81] Katrien Quintelier et al. “Analyzing high-dimensional cytometry data using FlowSOM”. en. In: *Nat. Protoc.* 16.8 (Aug. 2021), pp. 3775–3801. DOI: [10.1038/s41596-021-00550-0](https://doi.org/10.1038/s41596-021-00550-0).
- [82] Annelies Emmaneel et al. “PeacoQC: Peak-Based Selection of High Quality Cytometry Data”. en. In: *Cytometry A* (Sept. 2021). DOI: [10.1002/cyto.a.24501](https://doi.org/10.1002/cyto.a.24501).
- [83] Gianni Monaco et al. “flowAI: automatic and interactive anomaly discerning tools for flow cytometry data”. en. In: *Bioinformatics* 32.16 (Aug. 2016), pp. 2473–2480. DOI: [10.1093/bioinformatics/btw191](https://doi.org/10.1093/bioinformatics/btw191).
- [84] Justin Meskas et al. “flowCut: An R package for automated removal of outlier events and flagging of files based on time versus fluorescence analysis”. en. In: *Cytometry A* 103.1 (Jan. 2023), pp. 71–81. DOI: [10.1002/cyto.a.24670](https://doi.org/10.1002/cyto.a.24670).
- [85] Paulina Rybakowska et al. “Data processing workflow for large-scale immune monitoring studies by mass cytometry”. en. In: *Comput. Struct. Biotechnol. J.* 19 (May 2021), pp. 3160–3175. DOI: [10.1016/j.csbj.2021.05.032](https://doi.org/10.1016/j.csbj.2021.05.032).

- [86] Rachel Finck et al. “Normalization of mass cytometry data with bead standards”. en. In: *Cytometry A* 83.5 (May 2013), pp. 483–494. DOI: [10.1002/cyto.a.22271](https://doi.org/10.1002/cyto.a.22271).
- [87] Florian Hahne et al. *flowStats: Statistical methods for the analysis of flow cytometry data*. R package version 4.21.0. 2025. DOI: [10.18129/B9.bioc.flowStats](https://doi.org/10.18129/B9.bioc.flowStats). URL: <https://bioconductor.org/packages/flowStats>.
- [88] Matei Ionita et al. “Cleanet: robust doublet detection in cytometry data based on protein expression patterns”. en. In: *bioRxiv* (Jan. 2025), p. 2025.01.09.632259. DOI: [10.1101/2025.01.09.632259](https://doi.org/10.1101/2025.01.09.632259).
- [89] Mehrnoush Malek et al. “flowDensity: reproducing manual gating of flow cytometry data by automated density-based cell population identification”. en. In: *Bioinformatics* 31.4 (Feb. 2015), pp. 606–607. DOI: [10.1093/bioinformatics/btu677](https://doi.org/10.1093/bioinformatics/btu677).
- [90] Sofie Van Gassen et al. “CytoNorm: A Normalization Algorithm for Cytometry Data”. en. In: *Cytometry A* 97.3 (Mar. 2020), pp. 268–278. DOI: [10.1002/cyto.a.23904](https://doi.org/10.1002/cyto.a.23904).
- [91] Christina Bligaard Pedersen et al. “cyCombine allows for robust integration of single-cell cytometry datasets within and across technologies”. en. In: *Nat. Commun.* 13.1 (Mar. 2022), p. 1698. DOI: [10.1038/s41467-022-29383-5](https://doi.org/10.1038/s41467-022-29383-5).
- [92] Ronald P Schuyler et al. “Minimizing Batch Effects in Mass Cytometry Data”. en. In: *Front. Immunol.* 10 (Oct. 2019), p. 2367. DOI: [10.3389/fimmu.2019.02367](https://doi.org/10.3389/fimmu.2019.02367).
- [93] Marie Trussart et al. “Removing unwanted variation with CytofRUV to integrate multiple CyTOF datasets”. en. In: *Elife* 9 (Sept. 2020). DOI: [10.7554/eLife.59630](https://doi.org/10.7554/eLife.59630).
- [94] Ilya Korsunsky et al. “Fast, sensitive and accurate integration of single-cell data with Harmony”. en. In: *Nat. Methods* 16.12 (Dec. 2019), pp. 1289–1296. DOI: [10.1038/s41592-019-0619-0](https://doi.org/10.1038/s41592-019-0619-0).
- [95] Samuel Granjeaud et al. “CytoBatchNorm: an R package with Graphical Interface for Batch Effects Correction of Cytometry Data”. en. In: *Journal of Bioinformatics and Systems Biology* 7 (Dec. 2024), pp. 237–250. DOI: [10.26502/jbsb.5107095](https://doi.org/10.26502/jbsb.5107095).
- [96] Katrien L A Quintelier et al. “CytoNorm 2.0: A flexible normalization framework for cytometry data without requiring dedicated controls”. en. In: *Cytometry A* (Jan. 2025). DOI: [10.1002/cyto.a.24910](https://doi.org/10.1002/cyto.a.24910).

- [97] Mike Phuycharoen et al. “Integration of unpaired and heterogeneous clinical flow cytometry data”. en. In: *bioRxiv* (Dec. 2023), p. 2023.12.18.572157. DOI: [10.1101/2023.12.18.572157](https://doi.org/10.1101/2023.12.18.572157).
- [98] Dehua Peng, Zhipeng Gui, and Huayi Wu. “Interpreting the curse of dimensionality from distance concentration and manifold effect”. In: *arXiv [cs.LG]* (Dec. 2023). URL: <http://arxiv.org/abs/2401.00422>.
- [99] Karl Pearson. “On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: [10.1080/14786440109462720](https://doi.org/10.1080/14786440109462720).
- [100] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: <http://jmlr.org/papers/v9/vandermaaten08a.html>.
- [101] Leland McInnes, John Healy, and James Melville. “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”. In: *arXiv [stat.ML]* (Feb. 2018). URL: <http://arxiv.org/abs/1802.03426>.
- [102] Peng Liu et al. “Recent Advances in Computer-Assisted Algorithms for Cell Subtype Identification of Cytometry Data”. en. In: *Front Cell Dev Biol* 8 (Apr. 2020), p. 234. DOI: [10.3389/fcell.2020.00234](https://doi.org/10.3389/fcell.2020.00234).
- [103] Etienne Becht et al. “Dimensionality reduction for visualizing single-cell data using UMAP”. en. In: *Nat. Biotechnol.* (Dec. 2018). DOI: [10.1038/nbt.4314](https://doi.org/10.1038/nbt.4314).
- [104] Dmitry Kobak and George C Linderman. “Initialization is critical for preserving global data structure in both t-SNE and UMAP”. en. In: *Nat. Biotechnol.* 39.2 (Feb. 2021), pp. 156–157. DOI: [10.1038/s41587-020-00809-z](https://doi.org/10.1038/s41587-020-00809-z).
- [105] Andrew Draganov et al. “ActUp: Analyzing and Consolidating tSNE and UMAP”. In: *arXiv [cs.LG]* (May 2023). URL: <http://arxiv.org/abs/2305.07320>.
- [106] Sebastian Damrich et al. “From t-SNE to UMAP with contrastive learning”. In: *arXiv [cs.LG]* (June 2022). URL: <http://arxiv.org/abs/2206.01816>.
- [107] Jan Niklas Böhm, Philipp Berens, and Dmitry Kobak. “Attraction-repulsion spectrum in neighbor embeddings”. In: *J. Mach. Learn. Res.* 23.95 (2022), pp. 1–32. ISSN: 1532-4435.

- [108] John A Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer New York, 2007. ISBN: 9780387393506. DOI: [10.1007/978-0-387-39351-3](https://doi.org/10.1007/978-0-387-39351-3). URL: <http://dx.doi.org/10.1007/978-0-387-39351-3>.
- [109] Warren S Torgerson. “Multidimensional scaling: I. theory and method”. en. In: *Psychometrika* 17.4 (Dec. 1952), pp. 401–419. DOI: [10.1007/bf02288916](https://doi.org/10.1007/bf02288916).
- [110] M Roederer et al. “Probability binning comparison: a metric for quantitating univariate distribution differences”. en. In: *Cytometry* 45.1 (Sept. 2001), pp. 37–46. DOI: [10.1002/1097-0320\(20010901\)45:1<37::aid-cyto1142>3.0.co;2-e](https://doi.org/10.1002/1097-0320(20010901)45:1<37::aid-cyto1142>3.0.co;2-e).
- [111] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. “The Earth Mover’s Distance as a Metric for Image Retrieval”. In: *Int. J. Comput. Vis.* 40.2 (Nov. 2000), pp. 99–121. DOI: [10.1023/A:1026543900054](https://doi.org/10.1023/A:1026543900054).
- [112] Darya Y Orlova et al. “Earth Mover’s Distance (EMD): A True Metric for Comparing Biomarker Expression Levels in Cell Populations”. In: *PLOS ONE* 11.3 (2016), e0151859. DOI: [10.1371/journal.pone.0151859](https://doi.org/10.1371/journal.pone.0151859).
- [113] John A Lee and Michel Verleysen. “Scale-independent quality criteria for dimensionality reduction”. en. In: *Pattern Recognit. Lett.* 31.14 (Oct. 2010), pp. 2248–2257. DOI: [10.1016/j.patrec.2010.04.013](https://doi.org/10.1016/j.patrec.2010.04.013).
- [114] John A Lee and Michel Verleysen. *LELEC2870 - Dimensionality reduction and data visualization (UCLouvain Course Notes)*. 2021.
- [115] Jan De Leeuw and Patrick Mair. “Shepard Diagram”. en. In: *Wiley StatsRef: Statistics Reference Online*. Chichester, UK: John Wiley & Sons, Ltd, June 2015, pp. 1–3. DOI: [10.1002/9781118445112.stat06268.pub2](https://doi.org/10.1002/9781118445112.stat06268.pub2).
- [116] Philippe Hauchamps et al. “Visual Quality Control With CytoMDS, a Bioconductor Package for Low Dimensional Representation of Cytometry Sample Distances”. In: *Cytometry Part A* 107.3 (2025), pp. 177–186. DOI: [10.1002/cyto.a.24921](https://doi.org/10.1002/cyto.a.24921).
- [117] Janet Staats et al. “Guidelines for Gating Flow Cytometry Data for Immunological Assays”. en. In: *Methods Mol. Biol.* 2032 (2019), pp. 81–104. DOI: [10.1007/978-1-4939-9650-6_5](https://doi.org/10.1007/978-1-4939-9650-6_5).
- [118] Peng Shi et al. “Recent advances in regulatory immune cells: exploring the world beyond Tregs”. en. In: *Front. Immunol.* 16 (May 2025), p. 1530301. DOI: [10.3389/fimmu.2025.1530301](https://doi.org/10.3389/fimmu.2025.1530301).

- [119] Xiling Liu et al. “The immunosuppressive role of MDSCs in HCC: mechanisms and therapeutic opportunities”. en. In: *Cell Commun. Signal.* 23.1 (Mar. 2025), p. 155. DOI: [10.1186/s12964-025-02170-7](https://doi.org/10.1186/s12964-025-02170-7).
- [120] Annika M Bruger et al. “Chapter Ten - Protocol to assess the suppression of T-cell proliferation by human MDSC”. In: *Methods in Enzymology*. Ed. by Lorenzo Galluzzi and Nils-Petter Rudqvist. Vol. 632. Academic Press, Jan. 2020, pp. 155–192. DOI: [10.1016/bs.mie.2019.05.046](https://doi.org/10.1016/bs.mie.2019.05.046).
- [121] Sofie Van Gassen et al. “FlowSOM: Using self-organizing maps for visualization and interpretation of cytometry data”. en. In: *Cytometry A* 87.7 (July 2015), pp. 636–645. DOI: [10.1002/cyto.a.22625](https://doi.org/10.1002/cyto.a.22625).
- [122] Katrien Quintelier et al. “Analyzing high-dimensional cytometry data using FlowSOM”. en. In: *Nat. Protoc.* 16.8 (Aug. 2021), pp. 3775–3801. DOI: [10.1038/s41596-021-00550-0](https://doi.org/10.1038/s41596-021-00550-0).
- [123] T Kohonen. “The self-organizing map”. en. In: *Proc. IEEE Inst. Electr. Electron. Eng.* 78.9 (1990), pp. 1464–1480. DOI: [10.1109/5.58325](https://doi.org/10.1109/5.58325).
- [124] V Whitney and Kevin M. “Algorithm 422: minimal spanning tree [H]”. In: *Communications of the ACM* 15.4 (1972), pp. 273–274.
- [125] José Luis García-Lapresta and David Pérez-Román. “Consensus-Based Agglomerative Hierarchical Clustering”. In: *Fuzzy Sets, Rough Sets, Multisets and Clustering*. Ed. by Vicenç Torra, Anders Dahlbom, and Yasuo Narukawa. Cham: Springer International Publishing, 2017, pp. 123–135. ISBN: 978-3-319-47557-8. DOI: [10.1007/978-3-319-47557-8_8](https://doi.org/10.1007/978-3-319-47557-8_8).
- [126] Peng Liu et al. “Comprehensive evaluation and practical guideline of gating methods for high-dimensional cytometry data: manual gating, unsupervised clustering, and auto-gating”. In: *Briefings in Bioinformatics* 26.1 (Dec. 2024), bbae633. ISSN: 1477-4054. DOI: [10.1093/bib/bbae633](https://doi.org/10.1093/bib/bbae633).
- [127] Xiao Liu et al. “A comparison framework and guideline of clustering methods for mass cytometry data”. en. In: *Genome Biol.* 20.1 (Dec. 2019), p. 297. DOI: [10.1186/s13059-019-1917-7](https://doi.org/10.1186/s13059-019-1917-7).
- [128] Lukas M Weber and Mark D Robinson. “Comparison of clustering methods for high-dimensional single-cell flow and mass cytometry data”. en. In: *Cytometry A* 89.12 (Dec. 2016), pp. 1084–1096. DOI: [10.1002/cyto.a.23030](https://doi.org/10.1002/cyto.a.23030).
- [129] Peng Qiu et al. “Extracting a cellular hierarchy from high-dimensional cytometry data with SPADE”. en. In: *Nat. Biotechnol.* 29.10 (Oct. 2011), pp. 886–891. DOI: [10.1038/nbt.1991](https://doi.org/10.1038/nbt.1991).

- [130] Lukas Fisch et al. “GateNet: A novel neural network architecture for automated flow cytometry gating”. en. In: *Comput. Biol. Med.* 179.108820 (Sept. 2024), p. 108820. DOI: [10.1016/j.compbiomed.2024.108820](https://doi.org/10.1016/j.compbiomed.2024.108820).
- [131] Huamin Li et al. “Gating mass cytometry data by deep learning”. en. In: *Bioinformatics* 33.21 (Nov. 2017), pp. 3423–3430. DOI: [10.1093/bioinformatics/btx448](https://doi.org/10.1093/bioinformatics/btx448).
- [132] Daniel Commenges et al. “cytometree: A binary tree algorithm for automatic gating in cytometry analysis”. en. In: *Cytometry A* 93.11 (Nov. 2018), pp. 1132–1140. DOI: [10.1002/cyto.a.23601](https://doi.org/10.1002/cyto.a.23601).
- [133] Hao-Chih Lee et al. “Automated cell type discovery and classification through knowledge transfer”. en. In: *Bioinformatics* 33.11 (June 2017), pp. 1689–1695. DOI: [10.1093/bioinformatics/btx054](https://doi.org/10.1093/bioinformatics/btx054).
- [134] Disi Ji et al. “Bayesian Trees for Automated Cytometry Data Analysis”. en. In: *Machine Learning for Healthcare Conference*. PMLR, Nov. 2018, pp. 465–483.
- [135] Quentin Blampey et al. “A biology-driven deep generative model for cell-type annotation in cytometry”. en. In: *Brief. Bioinform.* 24.5 (Sept. 2023). DOI: [10.1093/bib/bbad260](https://doi.org/10.1093/bib/bbad260).
- [136] Eugénie Lohmann, Laurent Gorvel, and Samuel Granjeaud. “Build heatmaps of cell frequencies and marker intensities for correct interpretations”. In: *Bioinformatics* biorxiv;2025.02.10.637380v1 (Feb. 2025). URL: <https://www.biorxiv.org/content/10.1101/2025.02.10.637380v1.full.pdf>.
- [137] Aaron T. L. Lun, Stephanie C. Hicks, and Raphael Gottardo. *Orchestrating Single-Cell Analysis with Bioconductor*. CRC Press, 2021. DOI: [10.1201/9780367410786](https://doi.org/10.1201/9780367410786).
- [138] Lukas M Weber et al. “diffcyt: Differential discovery in high-dimensional cytometry via high-resolution clustering”. en. In: *Commun Biol* 2 (May 2019), p. 183. DOI: [10.1038/s42003-019-0415-5](https://doi.org/10.1038/s42003-019-0415-5).
- [139] Christof Seiler et al. “CytoGLMM: conditional differential analysis for flow and mass cytometry experiments”. en. In: *BMC Bioinformatics* 22.1 (Mar. 2021), p. 137. DOI: [10.1186/s12859-021-04067-x](https://doi.org/10.1186/s12859-021-04067-x).
- [140] Yvan Saeys, Sofie Van Gassen, and Bart N Lambrecht. “Computational flow cytometry: helping to make sense of high-dimensional immunology data”. en. In: *Nat. Rev. Immunol.* 16.7 (July 2016), pp. 449–462. DOI: [10.1038/nri.2016.56](https://doi.org/10.1038/nri.2016.56).

- [141] Sofie Van Gassen et al. “FloReMi: Flow density survival regression using minimal feature redundancy”. en. In: *Cytometry A* 89.1 (Jan. 2016), pp. 22–29. DOI: [10.1002/cyto.a.22734](https://doi.org/10.1002/cyto.a.22734).
- [142] Christophe Vanderaa. “A principled approach and standardised software for mass spectrometry based single-cell proteomics data analysis”. 2024. URL: <http://hdl.handle.net/2078.1/284721>.
- [143] Josef Spidlen et al. “Data file standard for flow cytometry, version FCS 3.2”. en. In: *Cytometry A* 99.1 (Jan. 2021), pp. 100–102. DOI: [10.1002/cyto.a.24225](https://doi.org/10.1002/cyto.a.24225).
- [144] Josef Spidlen et al. “FlowRepository: a resource of annotated flow cytometry datasets associated with peer-reviewed publications”. en. In: *Cytometry A* 81.9 (Sept. 2012), pp. 727–731. DOI: [10.1002/cyto.a.22106](https://doi.org/10.1002/cyto.a.22106).
- [145] Sanchita Bhattacharya et al. “ImmPort, toward repurposing of open access immunological assay data for translational and clinical research”. en. In: *Sci. Data* 5 (Feb. 2018), p. 180015. DOI: [10.1038/sdata.2018.15](https://doi.org/10.1038/sdata.2018.15).
- [146] Caspar J Van Lissa et al. “WORCS: A workflow for open reproducible code in science”. en. In: *Data Sci.* 4.1 (May 2021), pp. 29–49. DOI: [10.3233/ds-210031](https://doi.org/10.3233/ds-210031).
- [147] Prasad Patil, Roger D Peng, and Jeffrey T Leek. “A visual tool for defining reproducibility and replicability”. en. In: *Nat. Hum. Behav.* 3.7 (July 2019), pp. 650–652. DOI: [10.1038/s41562-019-0629-z](https://doi.org/10.1038/s41562-019-0629-z).
- [148] Nicolas P Rougier et al. “Sustainable computational science: the ReScience initiative”. en. In: *PeerJ Comput Sci* 3 (Dec. 2017), e142. DOI: [10.7717/peerj-cs.142](https://doi.org/10.7717/peerj-cs.142).
- [149] Robert C Gentleman et al. “Bioconductor: open software development for computational biology and bioinformatics”. en. In: *Genome Biol.* 5.10 (Sept. 2004), R80. DOI: [10.1186/gb-2004-5-10-r80](https://doi.org/10.1186/gb-2004-5-10-r80).
- [150] Wolfgang Huber et al. “Orchestrating high-throughput genomic analysis with Bioconductor”. en. In: *Nat. Methods* 12.2 (Feb. 2015), pp. 115–121.
- [151] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2023. URL: <https://www.R-project.org/>.

- [152] B Ellis et al. *flowCore: flowCore: Basic structures for flow cytometry data*. R package version 2.21.0. 2025. DOI: [10.18129/B9.bioc.flowCore](https://doi.org/10.18129/B9.bioc.flowCore). URL: <https://bioconductor.org/packages/flowCore>.
- [153] Greg Finak and Mike Jiang. *flowWorkspace: Infrastructure for representing and interacting with gated and ungated cytometry data sets*. R package version 4.21.0. 2025. DOI: [10.18129/B9.bioc.flowWorkspace](https://doi.org/10.18129/B9.bioc.flowWorkspace). URL: <https://bioconductor.org/packages/flowWorkspace>.
- [154] Greg Finak, Wenxin Jiang, and Raphael Gottardo. “CytoML for cross-platform cytometry data sharing”. en. In: *Cytometry A* 93.12 (Dec. 2018), pp. 1189–1196. DOI: [10.1002/cyto.a.23663](https://doi.org/10.1002/cyto.a.23663).
- [155] Florian Mair et al. “The end of gating? An introduction to automated analysis of high dimensional cytometry data: Highlights”. en. In: *Eur. J. Immunol.* 46.1 (Jan. 2016), pp. 34–43. DOI: [10.1002/eji.201545774](https://doi.org/10.1002/eji.201545774).
- [156] Rebecca Grant et al. “Understanding the contribution of operator measurement variability within Flow Cytometry data analysis for Quality Control of Cell and Gene Therapy manufacturing”. en. In: *Measurement (Lond.)* 150.106998 (Jan. 2020), p. 106998. DOI: [10.1016/j.measurement.2019.106998](https://doi.org/10.1016/j.measurement.2019.106998).
- [157] Sebastiano Montante and Ryan R Brinkman. “Flow cytometry data analysis: Recent tools and algorithms”. en. In: *Int. J. Lab. Hematol.* 41 Suppl 1 (May 2019), pp. 56–62. DOI: [10.1111/ijlh.13016](https://doi.org/10.1111/ijlh.13016).

Part II

Research projects

Chapter 7

CytoPipeline & CytoPipelineGUI: a Bioconductor R package suite for building and visualising automated pre-processing pipelines for flow cytometry data

7.1 Context

At the outset of our PhD, we began exploring cytometry data analysis pipelines, initially focusing on preprocessing. Our aim was to compare existing methods and potentially design a benchmarking study. However, we quickly encountered three interrelated challenges:

1. the vast number of available methods and pipeline combinations,
2. the lack of standardised tools for diagnostics and evaluation, and
3. the difficulty of managing our own evolving code across multiple pipeline iterations.

To address these issues, we developed *CytoPipeline*, a tool aimed at bioinformaticians for designing, evaluating, and comparing preprocessing pipelines

for cytometry data. It was released as part of Bioconductor version 3.17 in April 2023. To complement it, we later developed *CytoPipelineGUI*, a companion package designed for both bioinformaticians and end users, offering interactive visualisations to support pipeline evaluation and comparison. *CytoPipelineGUI* was released in Bioconductor version 3.18 in October 2023.

7.2 Article

Pre-print

Hauchamps, Philippe, Babak Bayat, Simon Delandre, Mehdi Hamrouni, Marie Toussaint, Stéphane T. Temmerman, Dan Lin, and Laurent Gatto. 2023. “CytoPipeline & CytoPipelineGUI: A Bioconductor R Package Suite for Building and Visualizing Automated Pre-Processing Pipelines for Flow Cytometry Data.” *bioRxiv*. DOI: 10.1101/2023.10.10.561699.

Peer-reviewed publication

Hauchamps, Philippe, Babak Bayat, Simon Delandre, Mehdi Hamrouni, Marie Toussaint, Stéphane T. Temmerman, Dan Lin, and Laurent Gatto. 2024. “CytoPipeline and CytoPipelineGUI: A Bioconductor R Package Suite for Building and Visualizing Automated Pre-Processing Pipelines for Flow Cytometry Data.” *BMC Bioinformatics* 25 (1): 80. DOI: 10.1186/s12859-024-05691-z.

The main text of the preprint is provided in the following pages, while the corresponding supplementary materials are provided in [Appendix B](#).

Errata

The following items are errors and typos noticed after the publication of the peer-reviewed article (page, figure and table numbers here refer to the preprint):

- In p.3, first paragraph, 7th line (‘ but they tend to focus on only one part of the pipeline, which is usually the downstream analysis.’), instead of ‘downstream analysis’, one should read ‘population identification’.
- In p.8, and in Figure 10 (p.15) and Figure 11 (p.16), ‘recall’ should be replaced by ‘Negative Predictive Value (NPV)’.
- In the legend of Figure 9 (p.14): instead of ‘positive decimals’, one should read ‘positive decades’.

CytoPipeline & CytoPipelineGUI: A Bioconductor R package suite for building and visualizing automated pre-processing pipelines for flow cytometry data

Philippe Hauchamps¹, Babak Bayat², Simon Delandre², Mehdi Hamrouni²,
Marie Toussaint², Stéphane Temmerman², Dan Lin², Laurent Gatto¹ ✉

✉ For correspondence:
laurent.gatto@uclouvain.be

Present address: Computational
Biology and Bioinformatics, de
Duve Institute, Université
Catholique de Louvain, Av.
Hippocrate 75, 1200 Brussels,
Belgium

Data availability: Raw flow
cytometry data files, as well as
manual gating information, are
available on Zenodo
(DOI:10.5281/zenodo.8425840).

Code availability: All code needed
to reproduce the results presented
in the current article is available on
the following GitHub repository:
[github.com/UCLouvain-CBIO/
2023-CytoPipeline-code](https://github.com/UCLouvain-CBIO/2023-CytoPipeline-code), of which a
release has been archived on
Zenodo
(DOI:10.5281/zenodo.8425840).

Funding: This work was funded by
GlaxoSmithKline Biologicals S.A.,
under a cooperative research and
development agreement between
GlaxoSmithKline Biologicals S.A.
and de Duve Institute (UCLouvain).

Competing interests: B.B., S.D.,
M.H., M.T., S.T., and D.L. are
employees of the GSK group of
Companies. B.B., S.D., M.T., S.T.,
and D.L. report ownership of GSK
shares. B.B., M.T., and S.T. are
listed as inventors on patent(s)
owned by the GSK group of
companies. P.H. is a student at the
de Duve Institute (UCLouvain) and
participates in a post graduate
studentship program at GSK.

¹Computational Biology and Bioinformatics, de Duve Institute, UCLouvain, Belgium;
²GSK, Rixensart, Belgium.

Abstract

Background: With the increase of the dimensionality in flow cytometry data over the past years, there is a growing need to replace or complement traditional manual analysis (i.e. iterative 2D gating) with automated data analysis pipelines. A crucial part of these pipelines consists of pre-processing and applying quality control filtering to the raw data, in order to use high quality events in the downstream analyses. This part can in turn be split into a number of elementary steps : signal compensation or unmixing, scale transformation, debris, doublets and dead cells removal, batch effect correction, etc. However, assembling and assessing the pre-processing part can be challenging for a number of reasons. First, each of the involved elementary steps can be implemented using various methods and R packages. Second, the order of the steps can have an impact on the downstream analysis results. Finally, each method typically comes with its specific, non standardized diagnostic and visualizations, making objective comparison difficult for the end user.

Results: Here, we present *CytoPipeline* and *CytoPipelineGUI*, two R packages to build, compare and assess pre-processing pipelines for flow cytometry data. To exemplify these new tools, we present the steps involved in designing a pre-processing pipeline on a real life dataset and demonstrate different visual assessment use cases. We also set up a benchmarking comparing two pre-processing pipelines differing by their quality control methods, and show how the package visualization utilities can provide crucial user insight into the obtained benchmark metrics.

Conclusion: *CytoPipeline* and *CytoPipelineGUI* are two Bioconductor R packages that help building, visualizing and assessing pre-processing pipelines for flow cytometry data. They increase productivity during pipeline development and testing, and complement benchmarking tools, by providing user intuitive insight into benchmarking results.

Introduction

With recent advances in flow cytometry technologies, it has become possible to measure up to 50 markers simultaneously for the same single cells (McKinnon, 2018). As an immediate benefit, scientists now have access to richer flow cytometry experimental data. However, these advances also come at a cost, i.e. a more complex data analysis task. Indeed, traditional 'manual gating' data analysis procedures, which proceed by iterative hierarchical 2D representations of the data guided by biological knowledge, are unable to thoroughly extract the signal of interest from such high-dimensional data (Saeys, Van Gassen, and Lambrecht, 2016). There is therefore a need to complement such manual, expert-based approaches with *computational flow cytometry*, i.e. a set of computational algorithms and methods for automated, reproducible and data-driven flow cytometry data analysis (Saeys, Van Gassen, and Lambrecht, 2016).

Those computational flow cytometry approaches translate into so-called data analysis pipelines. Examples of such automated pipelines have been published in the recent literature (e.g. Quintelier et al., 2021, Nowicka et al., 2017, Rybakowska et al., 2021, Ashhurst et al., 2022, Rybakowska et al., 2022). These consist of a series of data processing steps that are executed, one after the other, with the output of one step becoming the input of the next step. Schematically, for a typical flow cytometry data analysis, these numerous steps can usually be grouped into three big parts, coming after initial data sample acquisition:

- *data pre-processing and quality control*, which consists in both filtering undesirable and low quality events, and increasing the signal to noise ratio of the raw data, in order to feed the downstream steps with data of the highest possible quality;
- *population identification*, which aims at labelling the events with names of cell populations of interest;
- *downstream statistical analysis*, which can range from the simplest descriptive count/frequencies per population, to building complex prediction models for an outcome of interest, possibly for a high number of samples.

In what follows, we will mainly focus on the data pre-processing part, which can itself be split into several sub-tasks, or *steps* (Liechti et al., 2021): compensation, scale transformation, control (and possibly removal) of batch effects, control of signal stability in time (*QC in time*), filtering of undesirable events like debris, doublets and dead cells. All these steps are crucial to avoid that the subsequent analysis gets perturbed with erroneous signal (see e.g. Mazza et al., 2018 for compensation, Finak et al., 2010 for scale transformation, Emmaneel et al., 2022 for signal stability in time, and den Braanker, Bongenaar, and Lubberts, 2021 for other steps).

However, building good, automated pre-processing pipelines, suitable for the particular type of biological samples and biological question, can be challenging for a number of reasons. First, for each elementary step, there might exist a number of different computational methods, each of those having numerous parameters available to the user. For example, looking only into methods available on the Bioconductor project (Huber et al., 2015) for controlling the signal stability (*QC in time*), one finds at least four different methods available: *flowAI* (Monaco et al., 2016), *flowClean* (Fletez-Brant et al., 2016), *PeacoQC* (Emmaneel et al., 2022), *flowCut* (Meskas et al., 2023), and each of these methods comes with 7 to 11 different parameters. Second, the order of steps is not always set in stone, and applying different orders can lead to different outcomes, an effect coined *steps interaction*. These two facts lead to what we refer to as the *combinatorial problem of designing pipelines*, which means that, as the number of necessary elementary steps increases, the number of possible pipeline designs grows exponentially. As a consequence, for the user, it becomes time consuming to build and assess even only a few of the possible step combinations, let alone testing a representative sample of them in a systematic manner.

On top of that combinatorial problem, the user is also faced with a lack of generic, standardized

and user-friendly tools to evaluate and compare data pre-processing pipelines. On the one hand, each single pre-processing step method might come with its own approach for diagnostic and visualization (e.g. ad hoc plots, html or pdf reports), which allows the user neither to standardize the comparison process, nor to easily investigate the links and interactions between the different steps. On the other hand, there are a number of benchmarking studies comparing computational methods for flow cytometry data (Liu et al., 2019, Weber and Robinson, 2016, Cheung et al., 2022, Aghaeepour et al., 2016), but they tend to focus on only one part of the pipeline, which is usually the downstream analysis. Finally, there also exist some generic tools and frameworks to systematize the benchmarking process, including the interactions between different steps, such as for example *pipeComp* (Germain, Sonrel, and Robinson, 2020) and *CellBench* (Su et al., 2020), and as well as some attempts to formally model the pipeline optimization problem in mathematical terms (Selega and Campbell, 2022). However, what is lacking for the end user is the ability to intuitively interpret the results of such benchmarkings. In other words, could one translate that a pipeline *A* outperforms a pipeline *B*, with respect to a specific performance metric, in terms of the obtained data characteristics, or number of filtered events. Therefore, there is still a need for flow cytometry practitioner-focused standardized tools for visual comparison of pre-processing pipelines.

Here, we present *CytoPipeline* and *CytoPipelineGUI*, two *R* packages aimed at facilitating the design and visual comparison of pre-processing pipelines for flow cytometry data. We describe the concepts underlying the software, provide some illustrative examples and demonstrate the use of the accompanying visualization utilities. We show that these new tools can help increasing the productivity during pipeline development and testing, and that they can complement benchmarking tools and studies, by providing the user with intuitive insight into benchmarking results. The *CytoPipeline* and *CytoPipelineGUI* packages are available on Bioconductor (Huber et al., 2015), as of version 3.17 and 3.18, respectively.

Methods

Implementation

In what follows, we assume that we have a dataset, provided as a set of files in Flow Cytometry Standard (*fcs*) format (Spidlen et al., 2010), on which we would like to apply a data pre-processing pipeline.

The *CytoPipeline* suite is composed of two main *R* packages, *CytoPipeline* and *CytoPipelineGUI*. While *CytoPipeline* is the main package, providing support for pipeline definition, running, monitoring and basic plotting functions, *CytoPipelineGUI* provides two interactive GUI applications enabling users to interactively explore and visualize the pipeline results. The *CytoPipeline* framework is based on two main concepts, namely *CytoPipeline* and *CytoProcessingStep*. A *CytoPipeline* object centralizes the pipeline definition, and specifies the run order of the different pipeline steps. These steps materialize as *CytoProcessingStep* objects, which store pipeline step names and the corresponding *R* functions that will be called at execution time. These functions are either provided within the *CytoPipeline* package itself, exported from third party packages, or coded by the user themselves. Together with the function name to be called, a *CytoProcessingStep* object also contains the list of parameters that are used as arguments to the function.

When creating a *CytoPipeline* object, the user can provide the description of the pipeline as a text file in *json* format (Pezoa et al., 2016). Figure 1 shows the typical structure of such a *json* file. Note that, in practice, two different sets of processing steps, or pipelines, are described:

- A *scaleTransformProcessingSteps* pipeline, which describes the set of successive steps needed to generate the scale transformations that will be applied to the different channels of each

of the *fcs* files that are included in the dataset. The *CytoPipeline* engine will run this pipeline first, and only once, prior to running the pre-processing on each *fcs* file.

- A *flowFramesPreProcessingSteps* pipeline, which describes the set of pre-processing steps that will be applied on each of the different *fcs* file independently.

Steps in both pipelines are described in the exact same way, i.e. by providing a user-chosen name for the step, the corresponding function that needs to be called by the engine upon running, and the set of arguments (i.e. the list of parameter names and corresponding values) that need to be provided to the function. Note that, on top of these explicitly defined arguments, the running engine will also take the output of each step as an implicit additional argument to the function called by the subsequent step.

```
{
  "scaleTransformProcessingSteps": [
    {
      "name": ["my_scale_transform_step1"],
      "FUN": ["scaleTransformFunc1"],
      "ARGS": {
        "paramName1": ["value1"],
        "paramName2": ["value2"],
        "paramName3": ["value3"]
      }
    },
    {
      "name": ["my_scale_transform_step2"],
      "FUN": ["scaleTransformFunc2"],
      "ARGS": {
        "paramName4": ["value4"]
      }
    },
    (...) other scale transform processing steps
  ],
  "flowFramesPreProcessingSteps": [
    {
      "name": ["my_fcsfile_preprocessing_step1"],
      "FUN": ["preprocessingFunc1"],
      "ARGS": {
        "paramName5": ["value5"],
        "paramName6": ["value6"]
      }
    },
    {
      "name": ["my_fcsfile_preprocessing_step2"],
      "FUN": ["preprocessingFunc2"],
      "ARGS": []
    },
    (...) other flow frames preprocessing steps
  ]
}
```

Figure 1. Structure of the user provided *json* file that describes a *CytoPipeline* object.

The first pipeline (i.e. *"scaleTransformProcessingSteps"*) specifies how the preliminary calculation of the scale transformations is performed. Here only its first two steps are described. The first step, named *"my_scale_transform_step1"*, consists in calling the *"scaleTransformFunc1"* function, with 3 parameters (*"paramName1"*, *"paramName2"* and *"paramName3"* provided as arguments, taking the specified *"value1"*, *"value2"* and *"value3"* value respectively. The second step, named *"my_scale_transform_step2"*, calls the *"scaleTransformFunc2"*, with only one single parameter, i.e. *"paramName4"* taking *"value4"* as value. The second pipeline (i.e. *"flowFramesPreProcessingSteps"*) specifies the set of pre-processing steps performed on each data file independently. Here again, only the first two steps are described. The first one, named *"my_fcsfile_preprocessing_step1"* calls the *"preprocessingFunc1"* function with two parameters, while the second one, named *"my_fcsfile_preprocessing_step2"* calls the *"preprocessingFunc2"* function with no parameter (apart from the output of the previous step which is always used as an implicit additional argument).

The standard process for using *CytoPipeline* to build, run and inspect pre-processing pipelines is

the following:

- define the pipeline by specifying the different steps using a descriptive text file, in *json* format¹;
- run the pipeline, possibly for several data files in parallel, which involves writing and executing a short *R* script (see following sections);
- monitor the execution process thanks to a *CytoPipeline* provided workflow visualization utility;
- visualize and compare the results at different stages, using the *CytoPipelineGUI* interactive GUI applications.

In terms of technical infrastructure, the *CytoPipeline* package suite makes itself internal use of several technical *R* packages:

- *BiocParallel* (Morgan et al., 2023) enabling parallel pre-processing of *fcs* sample files;
- *BiocFileCache* (Shepherd and Morgan, 2023) enabling storage (i.e. *caching*) of all intermediary results for further inspection;
- *shiny* (Chang et al., 2023) for interactive visualizations.

Illustrative dataset

In order to demonstrate *CytoPipeline* functionalities, we make use of an illustrative dataset, the *HBV chronic mouse* dataset. This dataset was collected during a preclinical study aimed at assessing the effect of different therapeutic vaccine regimens on the immune response of Hepatitis B Virus transduced mice.

In this study, 56 male and female HLA.A2/DRB1 transgenic mice (transgenic for the human HLA-A2 and HLA-DRB1 molecules) were used. HLA.A2/DRB1 mice from groups 1, 2 and 4 were transduced at day 0 with adeno-associated virus serotype 2/8 (AAV2/8-HBV) vector carrying a replication-competent HBV DNA genome and randomized before immunization with 4 doses of vaccine candidate, based on level of HBs circulating antigen detected in the sera at day 21, age and gender proportions. Mice from group 3 were not transduced with AAV2/8-HBV viral vector and were immunized with four doses of vaccine candidate and finally, mice from group 5 were not transduced and received four doses of NaCl solution. Upon sacrifice, livers were collected, perfused with Phosphate Buffered Saline (PBS) to remove blood cells and after enzymatic treatment, lymphocytes were isolated, stained with different monoclonal antibodies. The stained cells were acquired by flow cytometry using a BD Symphony A5 flow cytometer - the same instrument for all biological samples - and analyzed using the FlowJo v10.8 Software (BD Life Sciences).

Animal husbandry and experimental procedures were ethically reviewed and carried out in accordance with European Directive 2010/63/EU and the GlaxoSmithKline Biologicals' policy on the care, welfare and treatment of animals, in GSK animal facilities located in Rixensart, Belgium (AAALAC accredited). The ethical protocol of the GSK in vivo study was approved by the local GSK ethical committee.

This experiment resulted in the acquisition of 55 different *fcs* raw data file - one sample could not be acquired - with a flow cytometry panel of 12 different channels. The *HBV chronic mouse* dataset is available on Zenodo (DOI:10.5281/zenodo.8425840).

¹ note that *CytoPipeline* also provides methods to define a pipeline and its steps programmatically in *R*, without providing a text file as an input.

Applied pre-processing pipelines

Pipeline set-up

For the purpose of illustrating *CytoPipeline* functionalities, the 55 raw data files of the *HBV chronic mouse* dataset were used as input of two different pre-processing pipelines. Each pipeline was composed of the following steps:

- Reading of the raw *fcs* sample files, using the *flowCore* package (Ellis et al., 2023).
- Margin events removal, which consists in identifying and removing the outliers using the *PeacoQC* package (Emmaneel et al., 2022). In short, manual boundaries per channel, corresponding to the instrument detection limits, are applied, and all events falling outside these boundaries are removed.
- Signal compensation, which consists in applying an existing compensation matrix. This matrix was generated by the flow cytometer at data acquisition time, and subsequently manually adjusted by the expert scientist.
- *QC in time*, which consists in eliminating parts of the signal that are not stable in time, using one of the corresponding QC algorithms (see below).
- Doublet removal, which consists in keeping the events that have a similar area vs. height ratio of the FSC channel signal pulse, and eliminating the doublets, which have a significantly higher ratio. This was performed using an ad hoc implementation in the *CytoPipeline* package.
- Debris removal, which consists in clustering the events in the (FSC-A, SSC-A) 2D representation, targeting a number of clusters provided by the user. After the clusters are obtained, the cluster of which the centroid lies nearest to the origin, i.e. with the smallest FSC-A (size) and smallest SSC-A (content, granularity), is considered as containing debris and removed. This was done using the *flowClust* package (Lo et al., 2009).
- Scale transformation, which consists in automatically estimating the parameters of a *logicle* transformation (Parks, Roederer, and Moore, 2006), using the *flowCore* package (Ellis et al., 2023). The obtained scale transformations were applied on all 55 sample files, and the parameters were estimated on an aggregation of a subset of 4 randomly chosen sample *fcs* files, after margin events removal and signal compensation.
- Dead cells removal, which consists in automatically setting a threshold between live cells and dead cells in the corresponding fluorescent 'Live & Dead' channel dimension, using the *flowDensity* package. The events having a 'Live & Dead' intensity above the found threshold are eliminated as dead cells.

However, the two pre-processing pipelines essentially differed by the method used for the *QC in time* step, as one used the *PeacoQC* package (Emmaneel et al., 2022), while the other used the *flowAI* package (Monaco et al., 2016). In addition, the step order was also different, as the *PeacoQC* method is based on a peak detection algorithm which needs to run on compensated, scaled transformed data (Emmaneel et al., 2022), while the *flowAI* method is advised to be applied on raw data (Monaco et al., 2016). Figure 2 outlines the different steps applied in the pre-processing of each *fcs* files, for both the *PeacoQC*-based pipeline, and the *flowAI*-based pipeline.

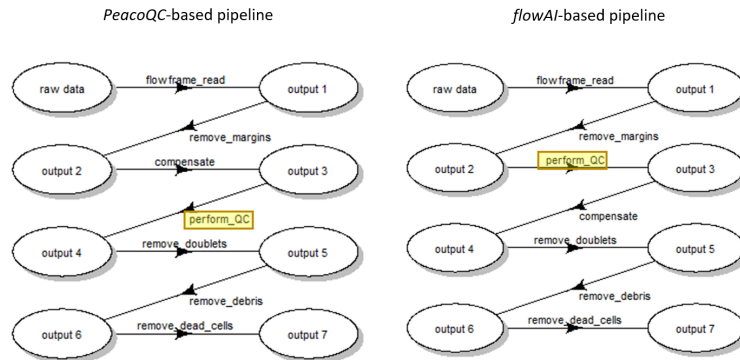


Figure 2. Workflow of the subsequent steps applied in the pre-processing of each *fcs* file, for both pipelines. Note that these plots have been generated using the *CytoPipeline* package.

More detailed information on packages, versions and methods underlying each step (Tables S1, S2 & S3), as well as the *json* configuration files defining respectively the *PeacoQC*-based and *flowAI*-based pipelines are available in the supplementary materials.

Running the pipelines and visualizing the results

In order to create the *CytoPipeline* objects representing the pipelines, run them and visualizing the results - including monitoring of the pipeline execution - a short *R* script needs to be written and executed. An example of such *R* script is provided in Figure S1. Note that, as a result of the centralization of the pipeline definition, the code is very simple and concise, as for example, creating and running the pipeline boils down to essentially two *R* statements. Also, note that it is the same *R* code that triggers the execution of both pre-processing pipelines described in the previous section (except for the selection of the appropriate input *json* file and the choice of the experiment name under which to store the results). The distinctive part of the pipeline is located in the input *json* file, which describes the pipelines steps and their execution order.

Example benchmarking

Aiming at illustrating the use of *CytoPipeline* to provide insights into benchmarking results, we designed a benchmarking, which consisted in comparing the outcome of the two competing *PeacoQC*-based and *flowAI*-based pipelines described in the previous section, using the *HBV chronic mouse* dataset, to a ground truth. The latter was obtained by submitting the 55 raw data *fcs* files to an expert scientist, who manually pre-processed the files, gated the events using FlowJo. The obtained FlowJo workspace file was subsequently automatically processed using the *CytoML* package (Finak, Jiang, and Gottardo, 2018) version 2.12.0, and incorporated into a dedicated *CytoPipeline* ground truth pipeline for comparison with the two automated pipelines.

Regarding the benchmark evaluation metrics, for each single *fcs* file, the final output of each pipeline was compared to the ground truth, in terms of number of events, and the following metrics were calculated: sensitivity, specificity, precision and recall, which are defined as follows:

let

- G (resp. B) be the set of events that are considered as **Good** (resp. **Bad**) in the manual gating i.e. in the ground truth;

- F_G (resp F_B) be the set of events that are flagged as good (resp. flagged as bad) by the considered automated pipeline.

We can additionally define the following sets of events:

- $F_{G, \text{correct}} = F_G \cap G$, the set of events that are correctly flagged as good;
- $F_{B, \text{correct}} = F_B \cap B$, the set of events that are correctly flagged as bad.

The chosen evaluation metrics are then defined as:

sensitivity = $\frac{|F_{B, \text{correct}}|}{|B|}$; specificity = $\frac{|F_{G, \text{correct}}|}{|G|}$; precision = $\frac{|F_{B, \text{correct}}|}{|F_B|}$; recall = $\frac{|F_{G, \text{correct}}|}{|F_G|}$,
where $|A|$ stands for the number of elements in the set A .

The benchmark was set up and performed using the *pipeComp* package (Germain, Sonrel, and Robinson, 2020), version 1.10.0. Indeed, *pipeComp* is a convenient tool to efficiently automate multiple alternative pipelines to be compared in the benchmark, as well as to automate the calculation of the evaluation metrics for each dataset used as benchmark input.

Results

Visual assessment and comparison of pipeline outputs

We used *CytoPipeline* to define both *PeacoQC*-based and *flowAI*-based pre-processing pipelines, as described in the Methods section, on the *HBV chronic mouse* dataset. We obtained results in the form of sets of data matrices (or *flowFrames*) after each step for each pre-processing pipeline. In the following paragraphs, we present some *CytoPipeline* visual assessment plots, according to 6 different use cases (Table 1). Use case #1 consists in visualizing a run and monitoring the status of the different steps. Use cases #2 to #5 consist in either looking at 'what happened' within a single pipeline for a single biological sample in isolation (use case #2), or comparing two different situations (flow frames) involving different pipelines (use cases #3 and #4), or involving different biological samples within the same pipeline (use case #5). Finally, use case #6 consists in assessing, and possibly modifying, the scale transformations obtained during a pipeline execution.

Use case	Description	Sample	Pipeline	Output (cf. Figure 2)	Compared sample	Compared pipeline	Compared output (cf. Figure 2)
#1	Monitoring a run	D91_G01	PeacoQC	All	Not applicable	Not applicable	Not applicable
#2	Visualizing the effect of a single pipeline step	D91_G01	PeacoQC	output 2 (before 'compensate')	D91_G01	PeacoQC	output 3 (after 'compensate')
#3	Comparing the outcome of a pipeline step with different parameter values	D91_G01	PeacoQC	output 6 (after 'remove_debris', run with 3 clusters)	D91_G01	PeacoQC	output 6 (after 'remove_debris', run with 2 clusters)
#4	Comparing two different methods for one or several step(s)	D91_A01	PeacoQC	output 7 (after 'remove_dead_cells')	D91_A01	flowAI	output 7 (after 'remove_dead_cells')
#5	Comparing two different biological samples	D91_A01	PeacoQC	output 7 (after 'remove_dead_cells')	D93_B05	PeacoQC	output 7 (after 'remove_dead_cells')
#6	Visualization and update of generate scale transformations	D91_G01	PeacoQC	Not applicable	Not applicable	Not applicable	Not applicable

Table 1. Use cases of visual assessment and comparison of pipeline outputs. When the use case involves comparing two flow frames obtained from different steps and/or different pipelines (i.e. use cases #2 to #4), or different samples (i.e. use case #5) the 3 columns 'sample', 'pipeline' and 'output' designate the initial flow frame (referring to Figure 2), while the 3 columns 'compared sample', 'compared pipeline' and 'compared output' designate the flow frame that is compared to the initial flow frame.

Use case #1: Monitoring a run

As all the intermediate results produced during pipeline execution are saved (see Methods/Implementation section), it is possible to generate a summary workflow view, consecutive to a run. Figure 3 shows an example of such a display, obtained after running the *flowAI*-based pipeline described above, where there was a spelling error in one of the parameter names of the "remove_debris" step. On top of showing the sequence of steps, a colour code is used to highlight which of the steps have run to completion, and which of the steps need to be re-run. Here, for the selected sample, the

pipeline ran correctly until the “*remove_doublets*” step (green nodes), but did not produce any output for the subsequent steps (orange nodes), which is due to the spelling error in the definition of the “*remove_debris*” step. Based on this summarized visual information, the user can now dig into the flagged problematic step, and/or track the particular characteristics of the sample which generated the error. More details on the colour code used in this plot can be found in Figure S2.

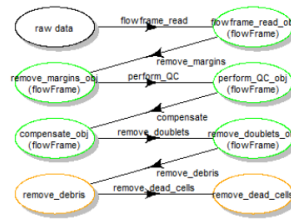


Figure 3. Use case #1: summary workflow view of the run - green nodes correspond to steps that ran to completion for the selected sample file, orange nodes correspond to steps that have not generated an output yet.

Use case #2: Visualizing the effect of a single pipeline step

In Figure 4, the user is visually assessing two consecutive states of the *flowFrame* of sample *D91_G01*, within the same run of the *PeacoQC*-based pipeline. To evaluate the effect of the compensation step, the "*before compensation*" (output 2, cf. Figure 2) and the "*after compensation*" (output 3) states of the pipeline are visually compared. Note that this visualization can be done according to any pair of selected channels/markers (2D distribution representation), or according to a 1D marginal distribution representation for any selected channel/marker. Here, the (CD8, CD38) 2D view shows, on the left, that the fluorescence of the dye BB700 (CD38) spills into the CD8 channel. On the right, application of a pre-computed compensation matrix (see Methods section) has rectified the distribution of the two markers, revealing different ranges of CD38 (an activation marker) between the CD8+ and CD8- populations. Note that a corresponding screenshot of the interactive GUI application, implemented in the *CytoPipelineGUI* package, can be found in Figure S3.

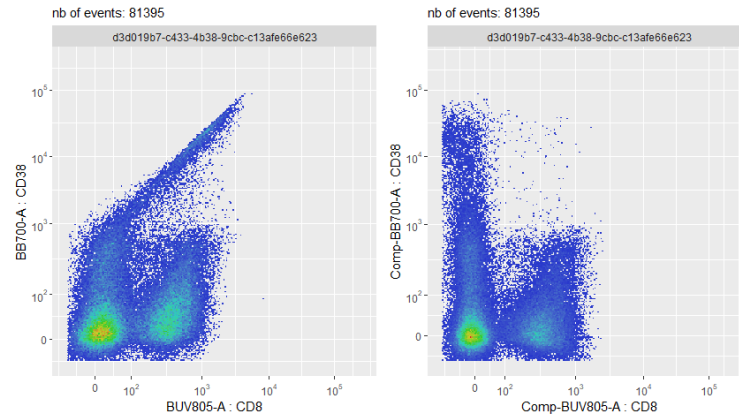


Figure 4. Use case #2: effect of a single pipeline step - here, the compensation step of the *PeacoQC*-based pipeline for sample *D91_G01*. On the left, spillover of the BB700 (CD38) dye fluorescence into the CD8 channel creates a visual artefact, with events wrongly flagged as double positive CD8+ CD38+. On the right, compensation has rectified the bivariate distribution of the two markers.

Use case #3: Comparing the outcome of a pipeline step with different parameter values. This use case involves running the same pipeline, with the same steps but with amended values for one or several steps, in order to investigate which parameter combination performs better. An illustrating example is shown in Figure 5, where the outcome of the debris removal step is compared when applying two different user input number of clusters (three on the left plot, vs. two on the middle plot). On the right plot, events coloured in red are the ones that are eliminated when applying the debris removal step when the number of clusters is two, but not eliminated when the number of clusters is three. Let us recall that the debris elimination step consists in clustering the events in a fixed number of clusters, followed by the elimination of the cluster nearest to the origin - see Methods section. Here, specifically, the user can conclude that the debris removal algorithm (based on *flowClust* package) does a better job selecting the target events when the appropriate number of target clusters is used, i.e. two clusters, as on the middle plot. This is because the cell population of interest, here a population of lymphocytes extracted from mice liver tissues, naturally groups into one single cluster in the (FSC-A, SSC-A) 2D representation. As a consequence, in this case, two is the optimal number of clusters (one cluster of debris, one cluster of lymphocytes). Figure S4 illustrates the removal of events during the debris removal step, for the 2 clusters and the 3 clusters cases.

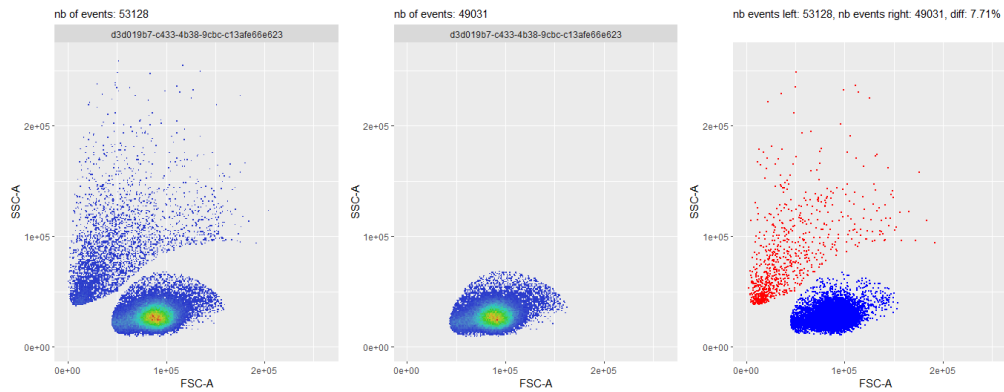


Figure 5. Use case #3: comparison of two different parameter settings for the debris removal step, on sample *D91_G01*. The setting with two clusters (in the middle) better eliminates undesirable events than the setting with three clusters (on the left). On the right, an explicit comparison between the two *flowFrames* is performed. Red dots correspond to events that are present on the left hand side plot, but not present on the middle plot, while blue dots correspond to events that are present on both plots.

Use case #4: Comparing two different methods for one or several steps

This use case is a generalization of the preceding one, where the user wants to compare the performance of two different methods for one or several steps of the pre-processing pipeline. For instance, Figure 6 provides a comparison between the *PeacoQC*-based and the *flowAI*-based pipelines, applied on a particular biological sample of the *HBV chronic mouse* dataset. This comparison, obtained by plotting one specific channel (here the *FSC-A*) as a function of time, reveals that *flowAI* removes time chunks more aggressively than *PeacoQC*, for the current sample. Note that this comparison can also be done for any 2D combination of makers (not shown here).

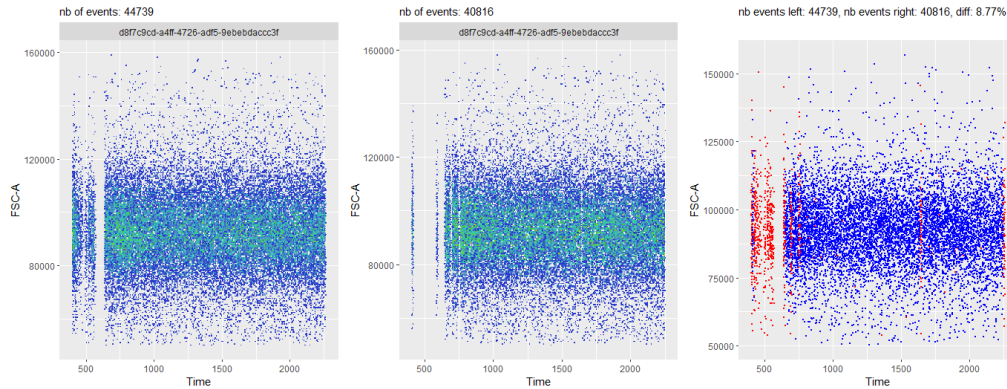


Figure 6. Use case #4: comparison of the final state results, for sample *D91_A01*, between the *PeacoQC*-based pipeline and the *flowAI*-based pipeline - here using a (*FSC-A* vs. *Time*) plot. On the left, the end state of the *PeacoQC*-based pipeline is shown, while the end state of the *flowAI*-based pipeline is shown on the middle plot. On the right, an explicit comparison between the two flowFrames is performed. Red dots correspond to events that are present on the left hand side plot, but not present on the middle plot, while blue dots correspond to events that are present on both plots. This figure reveals that, for this particular sample, *flowAI* tends to remove time chunks more aggressively than *PeacoQC*.

Use case #5: Comparing two different biological samples

It is also possible to compare two different biological samples of the same dataset, at any specific step of any pipeline. This allows e.g. to check that the methods used for the various pre-processing steps perform consistently across the whole dataset. One example is shown in Figure 7, where two different samples are displayed in a 2D plot with the *FSC-A* and *Live&Dead* channels. In this case, the two samples show very similar bivariate distributions. Based on this 2D representation, one could conclude that the pre-processing pipeline has correctly selected the target cell population in both cases. This would however need careful confirmation based on other 2D combinations of markers, e.g. *FSC-H* vs. *FSC-A* (for doublets elimination), and *FSC-A* vs. *SSC-A* (for debris elimination).

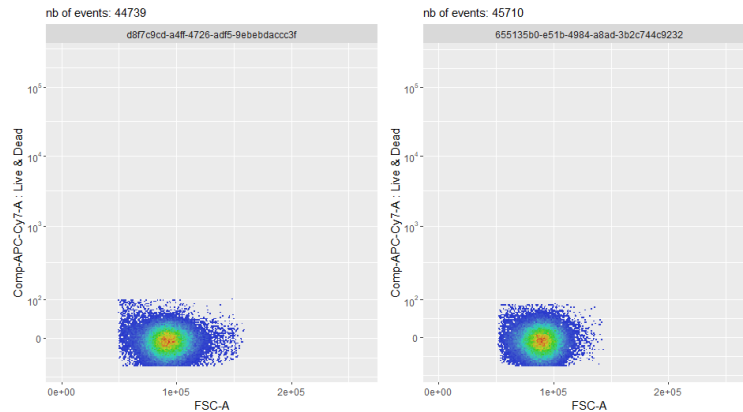


Figure 7. Use case #5: comparison of the final state results between two different biological samples (on the left: sample *D91_A01* and on the right: sample *D93_B05*), within the same *PeacoQC*-based pipeline. For the particular channels chosen (*Live&Dead* vs. *FSC-A*), the two samples show very similar bivariate distributions. Based on this 2D representation, one could conclude that the pre-processing pipeline has correctly selected the target cell population in both cases.

Use case #6: Visualization and update of generated scale transformations

Besides the *flowFrame* comparison tool, *CytoPipelineGUI* also provides a second interactive GUI application, which is aimed at inspecting the scale transformations obtained from the corresponding *scaleTransformProcessingSteps* pipeline (see Methods/Implementation section). If the shape of the distribution after transformation needs adjustment (for example for better separation of negative and positive populations for a specific marker), the user can manually adapt the scale transformation parameters, interactively assess the impact of their modifications, and apply these modifications to the scale transformations for further use in the pre-processing pipelines (Figure 8). These manual adjustments can be very useful, for example when the automatic transformation parameter adjustment algorithm has not worked satisfactorily. Figure 9 shows an example where the *logicle* transformation (Parks, Roederer, and Moore, 2006) applied on marker CD38 (left) shows spurious density oscillations in the negative domain. Manually adjusting the *positive decimals* parameter of the *logicle* transformation leads to a better looking density plot, where one can more easily distinguish CD38-, CD38+ and CD38++ populations.

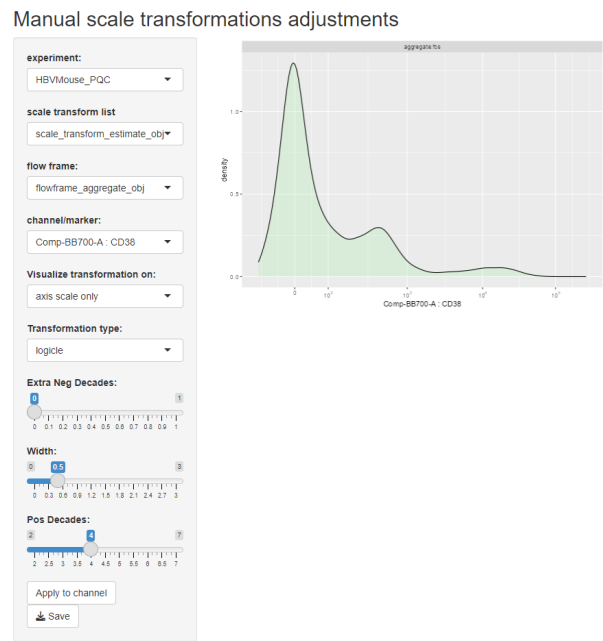


Figure 8. Use case #6: screenshot of the *CytoPipelineGUI* interactive GUI application enabling the inspection, manual adjustment and save of pipeline generated scale transformations. Here the user is visualizing the transformation applied on marker CD38, for sample *D91_G01*

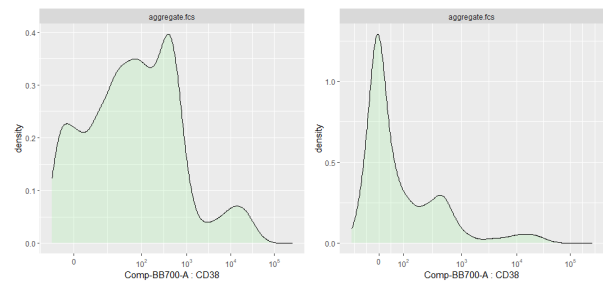


Figure 9. Manual parameters adjustment of the *logicle* transformation applied on marker CD38, for sample *D91_G01*. On the left, the density plot shows spurious oscillations in the negative domain. On the right, manual adjustment on the *positive decimals* parameter of the *logicle* transformation leads to a better looking transformed density, where one can more easily identify CD38-, CD38+ and CD38++ populations.

Benchmarking results

As mentioned in the Methods section, we use *pipeComp* (Germain, Sonrel, and Robinson, 2020) to perform a benchmarking exercise, comparing two different pre-processing pipelines, i.e. the *PeacoQC*-based and the *flowAI*-based pipelines, on the 55 sample files of the *HBV chronic mouse* dataset, and calculating evaluation metrics in terms of how well the automated pipelines could match the manual pre-processing performed by an expert scientist ('ground truth'). A global assessment shows comparable results between the two competing pipelines, consistently across all metrics (Figure 10). However, when directly contrasting sample by sample results (Figure 11) one can identify that the pipeline performance is rather heterogeneous across the 55 biological samples.

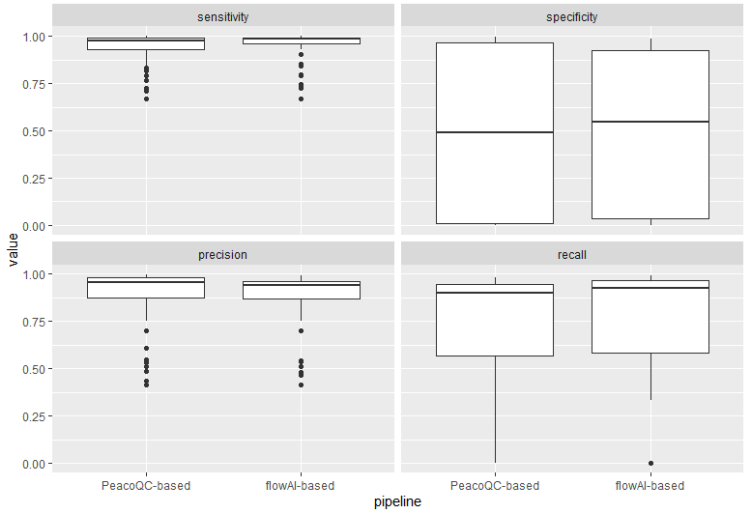


Figure 10. Box plots of the distributions of calculated evaluation metrics per sample, for the two competing pipelines. Globally, both pipelines perform very similarly, for all four evaluation metrics, i.e. sensitivity, specificity, precision and recall.

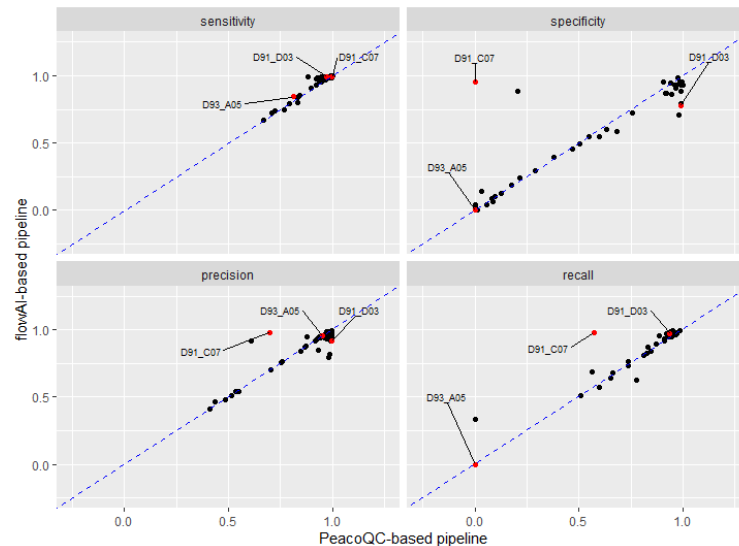


Figure 11. Scatter plots comparing the two pre-processing pipelines, each dot representing one of the 55 samples. Three specific samples are highlighted in red, corresponding to very different comparative behaviour of the two competing pipelines. Sample *D91_C07* is a unique sample for which the *flowAI*-based pipeline has a high specificity, but the *PeacoQC*-based pipeline has very low specificity. Sample *D93_A05* is one of the samples leading to low specificity for both pipelines, while sample *D91_D03* is representative of the samples for which both pipelines provide good specificity.

In order to better understand the behaviour of the two competing automatic pipelines on different samples, we selected three different samples, corresponding to different locations into the specificity plot of Figure 11. We then used *CytoPipelineGUI* to inspect the results at different steps, for the two automated pipelines as well as for the 'ground truth':

- Sample *D91_C07* was an outlier for which the *PeacoQC*-based pipeline obtained an almost zero specificity, while *flowAI*-based pipeline specificity was around an acceptable level of above 0.8. However, as shown in Figure 12, this was not due to the different *QC in time* algorithm (*PeacoQC* vs. *flowAI*), but to a lack of robustness of the dead cells removal algorithm, leading to an interaction phenomenon by which almost all events were removed in the dead cell removal step of the *PeacoQC*-based pipeline.
- Sample *D93_A05* resulted in a very low specificity for both pipelines. Investigation using *CytoPipelineGUI* revealed that this sample was in fact one of the low quality samples wherein the interesting cell population was a small minority of the events, while there was a great abundance of debris and dead cells (Figure S5). As a consequence, both pipelines were unable to automatically select the correct cell population, regardless of the *QC in time* method used.
- Sample *D91_D03* was an example where both automatic pipelines performed adequately without major issues. Here, the difference in metrics is effectively related to the choice of *QC in time* method. Looking at a specific visualization where time is displayed on the x axis (Figure S6), and based on both qualitative plot inspection and number of events comparison with the manual gating ground truth, *CytoPipelineGUI* reveals that *flowAI* method is too

aggressive in this case, while *PeacoQC* is too liberal.

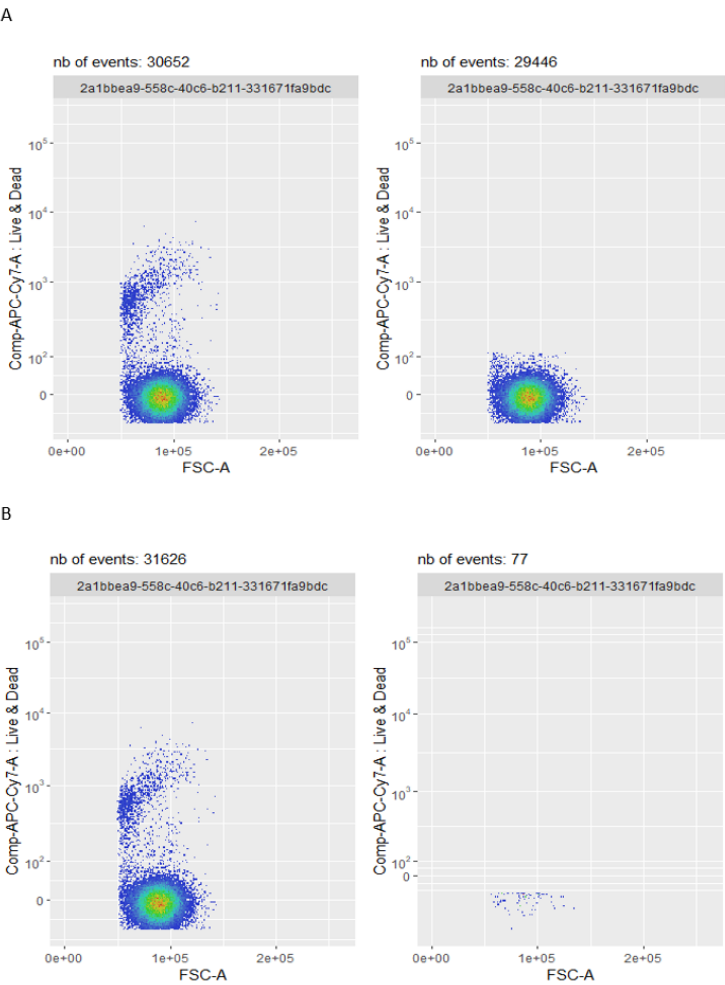


Figure 12. Comparison between the dead cells removal step between the *flowAI*-based pipeline (A), and the *PeacoQC*-based pipeline (B), on sample *D91_C07*. While the input set of events look very similar (left plots of panels A and B), the dead cells removal step of the *PeacoQC* pipeline (right plot of panel B) wrongly removes most of the events. This reveals a lack of robustness of the algorithm, unrelated to the *QC in time* method used (*flowAI* vs. *PeacoQC*).

Note that the conclusions of these visual inspections are particularly precious to the scientist in charge of building the data analysis pipelines, who is now able to get precise and accurate insight

into why one pipeline performs better than the other, for specific samples. In particular, they are much better equipped to distinguish between an intrinsic performance difference between some competing methods, and surprising artefacts like a side effect of low sample quality or an interaction between two different steps.

Discussion

CytoPipeline, a flexible framework for building and running pre-processing pipelines

In this work, we have demonstrated the use of the *CytoPipeline* suite by implementing pre-processing pipelines on the *HBV chronic mouse* dataset. The implementation of *CytoPipeline*, with a centralized specification of the pipeline definition in a *json* file, leads to a better design of the pipeline code. As a result, we believe that the user productivity, when coding and testing different pipelines, can be greatly improved.

In order to illustrate this, we implemented the two *PeacoQC*-based and *flowAI*-based competing pipelines, described in Methods, in two *R* scripts, without using *CytoPipeline* objects, and looked into the duplication effort as well as the future extensibility of the code. These pieces of code are provided in the *2023-CytoPipeline-code* GitHub repository (see Code Availability in Declarations section).

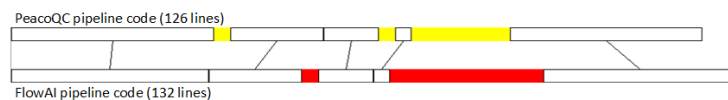


Figure 13. Structure of the *R* script implementations of the *PeacoQC*-based and *flowAI*-based pipelines. The common parts are shown in white, *PeacoQC*-based pipeline specific parts in yellow, and *flowAI*-based pipeline specific parts in red. Between the two pipelines, 79% of the code is in common, and the pipeline specific parts are not fully gathered in one single location.

Figure 13 provides a schematic comparison between these two pieces of code, as well as indicative number of code lines. Of course, these relates to one particular implementation, as there are countless ways to program the same pipelines. What is interesting to note, though, is that there is a high proportion of code duplication, but the differences are not only located in one single place, due to the subtle differences induced by the change of orders in the steps. This is likely to lead to a high code maintenance burden in the future, for instance when extending the program to many more pipeline instances, which can use different step methods, different method parameters etc. In contrast, let us recall that, when using *CytoPipeline*, the *R* code itself stays the same, as all differences are explicitly described in the input *json* file. This *json* file is easier to maintain and extend than the *R* scripts represented in Figure 13.

CytoPipeline provides a standardized and user-friendly tool for visual investigations

We have presented a series of use cases of *CytoPipeline* visualizations. In all these use cases, we took advantage of the same set of visualization tools, in a standardized way, but translated into different contexts, whatever the underlying methods used for the pre-processing pipelines. Also during the investigation of the benchmarking results, visual comparisons could be made with a ground truth manual gating, again using the same tools. Besides, the interactive GUI applications, implemented in *CytoPipelineGUI*, provide user interactivity and facilitate the investigation process.

As stated in the introduction, these visual assessments are extremely important for the scientists, as they provide a unique mean to:

- visually control for the quality of the data samples, and acquire insight on the corresponding sample variability;
- visually check the robustness of the methods used in a given pre-processing pipeline, including the adequacy of the chosen user input parameters;
- visually compare different pre-processing pipeline settings. This can range from comparing different possible choices of method for a particular step, to assessing which one of two or more competing pipelines, possibly mixing different step methods in different orders, is performing better for the considered dataset.

CytoPipeline allows user intuitive insight into benchmarking results

As part of this work, we have implemented a benchmarking comparing two competing pre-processing pipelines, with the main objective of showing the benefits of using *CytoPipeline* visualization tools, as a complement to the benchmarking itself. We showed that detailed comparison plots help the user investigating some specific benchmarking results, hence getting better intuition into the benchmarking outcome. We have indeed demonstrated that there can be numerous reasons why a pipeline instance performs better than another on specific samples, and it is key for the scientist to get a clear view of these reasons, and their possible links with sample characteristics. Therefore, we think that *CytoPipeline* is a powerful tool for interpreting the outcome of benchmarking studies.

Using the proportion of events kept at each step as a diagnostic tool

As was shown in various figures in the Results section (see e.g. Figure 12), *CytoPipelineGUI* computes the number of events that are retained at each step (shown as subtitles in the individual density plots). Tracking these changes throughout the pre-processing steps of a pipeline for different samples is a useful quality control. This can be implemented using some of the *CytoPipeline* functions, and is shown on Figure 14.

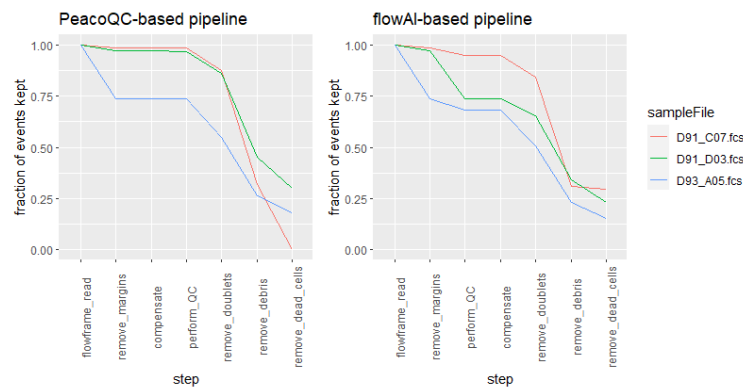


Figure 14. Plots showing the proportion of retained events at each pre-processing step, for each sample. On the left, the *PeacoQC*-based pipeline shows, for sample *D91_C07*, a sharp drop in the last *remove_dead_cells* step. On the right, the *flowAI*-based pipeline does not show the same phenomenon.

Limitations and possible extensions of the work

The *CytoPipeline* suite of R packages can be positioned as a tool to facilitate the design, testing and comparison of pre-processing pipelines for the end user. It is not meant to be:

- A novel pre-processing pipeline in itself, as it does not provide new methods for the various pre-processing steps (although it includes some functions calling some widely used methods), nor an innovative way to combine some of these.
- A tool facilitating benchmarking automation, like *pipeComp*. For example, unlike *pipeComp* (Germain, Sonrel, and Robinson, 2020), *CytoPipeline* does not provide any optimization solution to reduce the amount of CPU time and memory to run a potentially huge amount of (combinations of) possible pipelines. However, as mentioned before, *CytoPipeline* is used to facilitate the interpretation of results produced with benchmarking tools.

Regarding scalability, one should distinguish CPU and memory from hard drive storage requirements. CPU- and memory-wise, *CytoPipeline* has no particular issues when dealing with large number of samples, as long as each single fcs file can fully reside in memory. Indeed, as described in the Methods section, the engine that executes pre-processing pipelines supports both sequential and parallel file processing, and benefits from all multi-tasking scheduling options provided by the *BiocParallel* Morgan et al., 2023 package. However, storage-wise, caching data at each step leads to large storage needs when processing many files. Typically, when analysing datasets including hundreds of fcs files, with several millions of events, compared across several pipelines and many processing steps, storage needs can require several terabytes. In those cases, users of *CytoPipeline* will typically need to call on high capacity storage facilities.

Another limitation of our work is the following: while *CytoPipelineGUI* is a powerful visualization tool for exploring specific pipeline steps for one or two samples, it does not provide an overall quality control of all samples at once. In that sense, it would be of useful, especially for large datasets, to provide a global view of how samples differ at each pre-processing step. As mentioned above, one such diagnostic view can be obtained, by plotting the fraction of retained events at each pre-processing step (Figure 14). Another promising approach focuses on the visualisation of all samples at once to identify specific outliers Hauchamps and Gatto, 2024.

Finally, another possible extension would be to further develop *CytoPipeline*, as to not only include the building and assessment of pre-processing steps, but also include support for subsequent steps of the data analysis: batch correction, population identification, etc.

Conclusion

In this work, we have introduced a suite of R packages, *CytoPipeline* and *CytoPipelineGUI*, that helps building, visualizing and assessing pre-processing pipelines for flow cytometry data. We have demonstrated several use cases on a real life dataset, and highlighted several concrete benefits of these tools. For the new user, the packages come with ample documentation and tutorial videos, accessible through the package vignettes. We trust that using *CytoPipeline* will favour productivity in testing and assessing alternative data pre-processing pipelines, with the aim of designing good pre-processing and QC solutions for each particular context. The latter can be the specific type of biological sample, technology used (conventional flow cytometry, cytof, spectral flow cytometry), panel composition, experimental design etc., which in turn highly depend on the biological question at hand.

Declarations

Funding

This work was funded by GlaxoSmithKline Biologicals S.A., under a cooperative research and development agreement between GlaxoSmithKline Biologicals S.A. and de Duve Institute (UCLouvain).

Competing Interests

- B.B., S.D., M.H., M.T., S.T., and D.L. are employees of the GSK group of Companies.
- B.B., S.D., M.T., S.T., and D.L. report ownership of GSK shares.
- B.B., M.T., and S.T. are listed as inventors on patent(s) owned by the GSK group of companies.
- P.H. is a student at the de Duve Institute (UCLouvain) and participates in a post graduate studentship program at GSK.
- L.G. reports no competing interest.

Author contributions

- Conceptualization: P.H., L.G.;
- Methodology: P.H., L.G., D.L., S.D.;
- Software: P.H., L.G.;
- Data collection: B.B., M.H.;
- Writing - original draft: P.H., B.B.;
- Writing - review & editing: L.G., D.L., S.T., S.D., M.H., M.T.;
- Supervision: L.G., D.L., S.T., M.T.

Availability of data and materials

Raw flow cytometry data files, as well as the manual gating information considered as the ground truth for the benchmarking, are available on Zenodo (DOI:10.5281/zenodo.8425840).

Code availability

All code needed to reproduce the results presented in the current article is available on the following GitHub repository: <https://github.com/UCLouvain-CBIO/2023-CytoPipeline-code>, of which a release has been archived on Zenodo (DOI:10.5281/zenodo.8425840).

Acknowledgment

This preprint was created using the LaPreprint template (<https://github.com/roaldarbol/lapreprint>) by Mikkel Roald-Arbøl .

References

- Aghaeepour, Nima, Pratip Chattopadhyay, Maria Chikina, Tom Dhaene, Sofie Van Gassen, Miron Kursa, Bart N Lambrecht, Mehrnosh Malek, G J McLachlan, Yu Qian, Peng Qiu, Yvan Saeys, Rick Stanton, Dong Tong, Celine Vens, Sławomir Walkowiak, Kui Wang, Greg Finak, Raphael Gottardo, Tim Mosmann, Garry P Nolan, Richard H Scheuermann, and Ryan R Brinkman (2016). "A benchmark for evaluation of algorithms for identification of cellular correlates of clinical outcomes". In: *Cytometry A* 89.1, pp. 16–21.
- Ashhurst, Thomas Myles, Felix Marsh-Wakefield, Givanna Haryono Putri, Alanna Gabrielle Spiteri, Diana Shinko, Mark Norman Read, Adrian Lloyd Smith, and Nicholas Jonathan Cole King (2022). "Integration, exploration, and analysis of high-dimensional single-cell cytometry data using Spectre". In: *Cytometry A* 101.3, pp. 237–253.
- Chang, Winston, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges (2023). *shiny: Web Application Framework for R*. URL: <https://shiny.posit.co/>.
- Cheung, Melissa, Jonathan J Campbell, Robert J Thomas, Julian Braybrook, and Jon Petzing (2022). "Assessment of automated flow cytometry data analysis tools within cell and gene therapy manufacturing". In: *Int. J. Mol. Sci.* 23.6, p. 3224.
- den Braanker, Hannah, Margot Bongenaar, and Erik Lubberts (2021). "How to Prepare Spectral Flow Cytometry Datasets for High Dimensional Data Analysis: A Practical Workflow". In: *Front. Immunol.* 12, p. 768113.

- Ellis, B, Perry Haaland, Florian Hahne, Nolwenn Le Meur, Nishant Gopalakrishnan, Josef Špidlen, Mike Jiang, and Greg Finak (2023). *flowCore: Basic structures for flow cytometry data*. R package version 2.12.0. DOI: [10.18129/B9.bioc.flowCore](https://doi.org/10.18129/B9.bioc.flowCore). URL: <https://bioconductor.org/packages/flowCore>.
- Emmaneel, Annelies, Katrien Quintelier, Dorine Sichien, Paulina Rybakowska, Concepción Maraño, Marta E Alarcón-Riquelme, Gert Van Isterdael, Sofie Van Gassen, and Yvan Saeys (2022). "Pea-coQC: Peak-based selection of high quality cytometry data". In: *Cytometry A* 101.4, pp. 325–338.
- Finak, Greg, Wenxin Jiang, and Raphael Gottardo (2018). "CytoML for cross-platform cytometry data sharing". In: *Cytometry A* 93.12, pp. 1189–1196.
- Finak, Greg, Juan-Manuel Perez, Andrew Weng, and Raphael Gottardo (2010). "Optimizing transformations for automated, high throughput analysis of flow cytometry data". In: *BMC Bioinformatics* 11, p. 546.
- Fletez-Brant, Kipper, Josef Špidlen, Ryan R Brinkman, Mario Roederer, and Pratip K Chattopadhyay (2016). "flowClean: Automated identification and removal of fluorescence anomalies in flow cytometry data". In: *Cytometry A* 89.5, pp. 461–471.
- Germain, Pierre-Luc, Anthony Sonrel, and Mark D Robinson (2020). "pipeComp, a general framework for the evaluation of computational pipelines, reveals performant single cell RNA-seq pre-processing tools". In: *Genome Biol.* 21.1, p. 227.
- Hauchamps, Philippe and Laurent Gatto (2024). *CytoMDS: Low Dimensions projection of cytometry samples*. R package version 0.99.8. URL: <https://uclouvain-cbio.github.io/CytoMDS>.
- Huber, Wolfgang, Vincent J Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S Carvalho, Hector Corrada Bravo, Sean Davis, Laurent Gatto, Thomas Girke, Raphael Gottardo, Florian Hahne, Kasper D Hansen, Rafael A Irizarry, Michael Lawrence, Michael I Love, James MacDonald, Valerie Obenchain, Andrzej K Oleś, Hervé Pagès, Alejandro Reyes, Paul Shannon, Gordon K Smyth, Dan Tenenbaum, Levi Waldron, and Martin Morgan (2015). "Orchestrating high-throughput genomic analysis with Bioconductor". In: *Nat. Methods* 12.2, pp. 115–121.
- Liechti, Thomas, Lukas M Weber, Thomas M Ashhurst, Natalie Stanley, Martin Prlic, Sofie Van Gassen, and Florian Mair (2021). "An updated guide for the perplexed: cytometry in the high-dimensional era". In: *Nat. Immunol.* 22.10, pp. 1190–1197.
- Liu, Xiao, Weichen Song, Brandon Y Wong, Ting Zhang, Shunying Yu, Guan Ning Lin, and Xianting Ding (2019). "A comparison framework and guideline of clustering methods for mass cytometry data". In: *Genome Biol.* 20.1, p. 297.
- Lo, Kenneth, Florian Hahne, Ryan R Brinkman, and Raphael Gottardo (2009). "flowClust: a Bioconductor package for automated gating of flow cytometry data". In: *BMC Bioinformatics* 10, p. 145.
- Mazza, Emilia Maria Cristina, Jolanda Brummelman, Giorgia Alvisi, Alessandra Roberto, Federica De Paoli, Veronica Zanon, Federico Colombo, Mario Roederer, and Enrico Lugli (2018). "Background fluorescence and spreading error are major contributors of variability in high-dimensional flow cytometry data visualization by t-distributed stochastic neighboring embedding". In: *Cytometry A* 93.8, pp. 785–792.
- McKinnon, Katherine M (2018). "Flow Cytometry: An Overview". In: *Curr. Protoc. Immunol.* 120, pp. 5.1.1–5.1.11.
- Meskas, Justin, Daniel Yokosawa, Sherrie Wang, Gabriela C Segat, and Ryan Remy Brinkman (2023). "flowCut: An R package for automated removal of outlier events and flagging of files based on time versus fluorescence analysis". In: *Cytometry A* 103.1, pp. 71–81.
- Monaco, Gianni, Hao Chen, Michael Poidinger, Jinmiao Chen, João Pedro de Magalhães, and Anis Larbi (2016). "flowAI: automatic and interactive anomaly discerning tools for flow cytometry data". In: *Bioinformatics* 32.16, pp. 2473–2480.
- Morgan, Martin, Jiefei Wang, Valerie Obenchain, Michel Lang, Ryan Thompson, and Nitesh Turaga (2023). *BiocParallel: Bioconductor facilities for parallel evaluation*. R package version 1.34.0. DOI: [10.18129/B9.bioc.BiocParallel](https://doi.org/10.18129/B9.bioc.BiocParallel). URL: <https://bioconductor.org/packages/BiocParallel>.

- Nowicka, Malgorzata, Carsten Krieg, Helena L Crowell, Lukas M Weber, Felix J Hartmann, Silvia Guglietta, Burkhard Becher, Mitchell P Levesque, and Mark D Robinson (2017). "CyTOF workflow: differential discovery in high-throughput high-dimensional cytometry datasets". In: *F1000Res*. 6, p. 748.
- Parks, David R, Mario Roederer, and Wayne A Moore (2006). "A new "Logicle" display method avoids deceptive effects of logarithmic scaling for low signals and compensated data". In: *Cytometry A* 69.6, pp. 541–551.
- Pezoa, Felipe, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč (2016). "Foundations of JSON schema". In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, pp. 263–273.
- Quintelier, Katrien, Artuur Couckuyt, Annelies Emmaneel, Joachim Aerts, Yvan Saeys, and Sofie Van Gassen (2021). "Analyzing high-dimensional cytometry data using FlowSOM". In: *Nat. Protoc.* 16.8, pp. 3775–3801.
- Rybakowska, Paulina, Sofie Van Gassen, Jordi Martorell Marugán, Katrien Quintelier, Yvan Saeys, Marta E Alarcón-Riquelme, and Concepción Marañón (2022). "Protocol for large scale whole blood immune monitoring by mass cytometry and Cyto Quality Pipeline". In: *STAR Protoc.* 3.4, p. 101697.
- Rybakowska, Paulina, Sofie Van Gassen, Katrien Quintelier, Yvan Saeys, Marta E Alarcón-Riquelme, and Concepción Marañón (2021). "Data processing workflow for large-scale immune monitoring studies by mass cytometry". In: *Comput. Struct. Biotechnol. J.* 19, pp. 3160–3175.
- Saeys, Yvan, Sofie Van Gassen, and Bart N Lambrecht (2016). "Computational flow cytometry: helping to make sense of high-dimensional immunology data". In: *Nat. Rev. Immunol.* 16.7, pp. 449–462.
- Selega, Alina and Kieran R. Campbell (2022). "Multi-objective Bayesian Optimization with Heuristic Objectives for Biomedical and Molecular Data Analysis Workflows". In: *bioRxiv*. DOI: [10.1101/2022.06.08.495370](https://doi.org/10.1101/2022.06.08.495370). eprint: <https://www.biorxiv.org/content/early/2022/06/12/2022.06.08.495370.full.pdf>. URL: <https://www.biorxiv.org/content/early/2022/06/12/2022.06.08.495370>.
- Shepherd, Lori and Martin Morgan (2023). *BiocFileCache: Manage Files Across Sessions*. R package version 2.8.0. DOI: [10.18129/B9.bioc.BiocFileCache](https://doi.org/10.18129/B9.bioc.BiocFileCache). URL: <https://bioconductor.org/packages/BiocFileCache>.
- Spidlen, Josef, Wayne Moore, David Parks, Michael Goldberg, Chris Bray, Pierre Bierre, Peter Gorombey, Bill Hyun, Mark Hubbard, Simon Lange, Ray Lefebvre, Robert Leif, David Novo, Leo Ostruszka, Adam Treister, James Wood, Robert F Murphy, Mario Roederer, Damir Sudar, Robert Zigon, and Ryan R Brinkman (2010). "Data file standard for flow cytometry, version FCS 3.1". In: *Cytometry A* 77.1, pp. 97–100.
- Su, Shian, Luyi Tian, Xueyi Dong, Peter F Hickey, Saskia Freytag, and Matthew E Ritchie (2020). "Cell-Bench: R/Bioconductor software for comparing single-cell RNA-seq analysis methods". In: *Bioinformatics* 36.7, pp. 2288–2290.
- Weber, Lukas M and Mark D Robinson (2016). "Comparison of clustering methods for high-dimensional single-cell flow and mass cytometry data". In: *Cytometry A* 89.12, pp. 1084–1096.

Chapter 8

Visual quality control with CytoMDS, a Bioconductor package for low dimensional representation of cytometry sample distances

8.1 Context

While exploring various cytometry data analysis pipelines, we observed that sample quality issues could still emerge in downstream analyses, despite standard preprocessing and cleaning steps. Besides, thorough inspection of marker intensity distributions across all samples proved to be both tedious and time-consuming.

To support a more efficient and holistic [QC](#) process, we aimed to develop a visual summary method that displays all samples simultaneously in a low dimensional space, with interpretable distances to highlight potential anomalies. This visualisation was meant as a complement to conventional per-sample ‘[QC](#) in time’ approaches.

With this goal, we developed the *CytoMDS* R package, which computes a [MDS](#) projection based on a matrix of [EMDs](#) between samples. Special attention was given to tools enabling both evaluation of the projection quality and user interpretation. *CytoMDS* was released in Bioconductor version 3.19

in April 2024.

8.2 Article

Pre-print

Hauchamps, Philippe, Simon Delandre, Stéphane T. Temmerman, Dan Lin, and Laurent Gatto. 2024. “Visual Quality Control with CytoMDS, a Bio-conductor Package for Low Dimensional Representation of Cytometry Sample Distances.” *bioRxiv*. DOI: 10.1101/2024.07.01.601465.

Peer-reviewed publication

Hauchamps, Philippe, Simon Delandre, Stéphane T. Temmerman, Dan Lin, and Laurent Gatto. 2025. “Visual Quality Control with CytoMDS, a Bio-conductor Package for Low Dimensional Representation of Cytometry Sample Distances.” *Cytometry. Part A: The Journal of the International Society for Analytical Cytology* 107 (3): 177–86. DOI: 10.1002/cyto.a.24921.

The main text of the preprint is provided in the following pages, while the corresponding supplementary materials are provided in Appendix [C](#).

Visual quality control with *CytoMDS*, a Bioconductor package for low dimensional representation of cytometry sample distances

Philippe Hauchamps¹, Simon Delandre², Stéphane T. Temmerman², Dan Lin², Laurent Gatto¹ ✉

¹Computational Biology and Bioinformatics, de Duve Institute, UCLouvain, Belgium;

²GSK, Rixensart, Belgium.

✉ For correspondence:
laurent.gatto@uclouvain.be

Present address: Computational Biology and Bioinformatics, de Duve Institute, UCLouvain, Av. Hippocrate 75, 1200 Brussels, Belgium

Data availability: Raw cytometry data files are available either on Zenodo (DOI:10.5281/zenodo.10572228) or FlowRepository (ID: FR-FCM-ZYL8). The *ImmunoSenescence Human PBMC* dataset is available upon request.

Code availability: All code needed to reproduce the results presented in the current article is available on the following GitHub repository: github.com/UCLouvain-CBIO/2024-CytoMDS-code.

Funding: This work was funded by GlaxoSmithKline Biologicals S.A., under a cooperative research and development agreement between GlaxoSmithKline Biologicals S.A. and de Duve Institute (UCLouvain).

Competing interests: S.D., S.T., and D.L. are employees of the GSK group of companies, and report ownership of GSK shares. S.T. is listed as inventor on patents owned by the GSK group of companies. P.H. is a student at the de Duve Institute (UCLouvain) and participates in a post graduate studentship program at GSK.

Abstract

Quality Control (QC) of samples is an essential preliminary step in cytometry data analysis. Notably, identification of potential batch effects and outlying samples is paramount, to avoid mistaking these effects for true biological signal in downstream analyses. However, this task can prove to be delicate and tedious, especially for datasets with dozens of samples.

Here, we present *CytoMDS*, a Bioconductor package implementing a dedicated method for low dimensional representation of cytometry samples composed of marker expressions for up to millions of single cells. This method allows a global representation of all samples of a study, with one single point per sample, in such a way that projected distances can be visually interpreted. *CytoMDS* uses *Earth Mover's Distance* for assessing dissimilarities between multi-dimensional distributions of marker expression, and *Multi Dimensional Scaling* for low dimensional projection of distances. Some additional visualization tools, both for projection quality diagnosis and for user interpretation of the projection coordinates, are also provided in the package.

We demonstrate the strengths and advantages of *CytoMDS* for QC of cytometry data on three real biological datasets, revealing the presence of low quality samples, batch effects and biological signal between sample groups.

Introduction

Recent technology advances in flow and mass cytometry allow the acquisition of high dimensional sample data, with up to respectively 40 and 50 parameters measured simultaneously for millions of cells (Kare et al., 2023, Spitzer and Nolan, 2016). For such high dimensional data samples, extracting relevant biological information becomes challenging (Newell and Cheng, 2016). Moreover, in large clinical studies, possibly with some longitudinal and multicentric components, this difficult task is itself multiplied by the high number of samples (Hartmann et al., 2019, Schmit, Klomp, and Khan, 2021). In the last decade, this led to the development of (semi) automated data analysis pipelines (Saey, Van Gassen, and Lambrecht, 2016), to complement the traditional manual gating

approaches (Staats et al., 2019). Among the different steps involved in (automated) data analysis pipelines, pre-processing and Quality Control (QC) of data samples is paramount, as it guarantees that the downstream analyses are not compromised by confounding noise. For cytometry data, pre-processing and QC involve different treatments and checks (Liechti et al., 2021): compensation, stability of signal in time, removal of undesirable events like debris, doublets or dead cells, batch effect detection and removal, etc.

Whatever the level of automation of these treatments, some manual checks of the cleaned samples, possibly at different stages of the pre-processing, are still necessary. This is typically done by human operators using interactive visual software, like e.g. FlowJo or Cytobank - among the available commercial platforms, or free open source solutions (Hammill, 2021; Hauchamps et al., 2024). Nevertheless, these tools are mostly designed to check samples one by one, or possibly to visually compare two or a handful of samples at once. However, two categories of QC, namely outlying sample identification (Chen et al., 2020) and batch effect detection (Goh, Wang, and Wong, 2017), would benefit from a global representation of the dataset. Well-known classical data exploration techniques like *Principal Component Analysis (PCA)* (Pearson, 1901) have the potential to provide such a global view of the data, and are routinely used in other omics context like e.g. bulk transcriptomics (Chen et al., 2020) and bulk proteomics (Goh and Wong, 2017) data analyses. In bulk experiments, each sample results in a single vector of measurements because all its cells are merged. All samples can be combined into a single matrix which corresponds to the use of a standard PCA, or a similar algorithm. In single cell experiments, there is a measurement vector for each cell and each sample results in a measurement matrix. Applying these bulk techniques to cytometry data presents a major challenge in summarising each sample by a vector for comparing samples (not their cells) with each other.

To address this challenge, we developed *CytoMDS* (Hauchamps and Gatto, 2024), an R package for low dimensional projection of cytometry samples, which allows a global representation of all samples of a study, with one single point per sample, in such a way that projected distances can be visually interpreted. In this article, we present the *CytoMDS* method, and demonstrate its use on three real cytometry datasets, illustrating the detection of various quality issues. Finally we compare our approach with similar existing methods for global problem detection in cytometry samples, and discuss its advantages and potential limitations. *CytoMDS* is available in Bioconductor (Huber et al., 2015), as of version 3.19.

Materials and Methods

Proposed method

Figure 1 sketches an overview of the *CytoMDS* workflow. As input, the method requires a set of minimally pre-processed - i.e. scale transformed and, if applicable, compensated or unmixed - sample files in *flow cytometry standard (fcs)* format (Figure 1, [A]). The *CytoMDS* method itself (Figure 1, [B]) can be decomposed into two main steps:

- computing the distances between each sample pair (1);
- projecting the resulting pairwise distance matrix onto a low dimensional space (2) of dimension p , p being typically 2, 3 or a small number sufficient to reach a good projection quality (see below).

As a result, each sample is a single point in this p -dimensional space, and all samples can be represented in plots combining those p dimensions. Optionally, *bi-plots* can be performed by overlaying on these plots the correlations (4) of chosen sample statistics (3) with the projection coordinates, in order to interpret the directions within the plots.

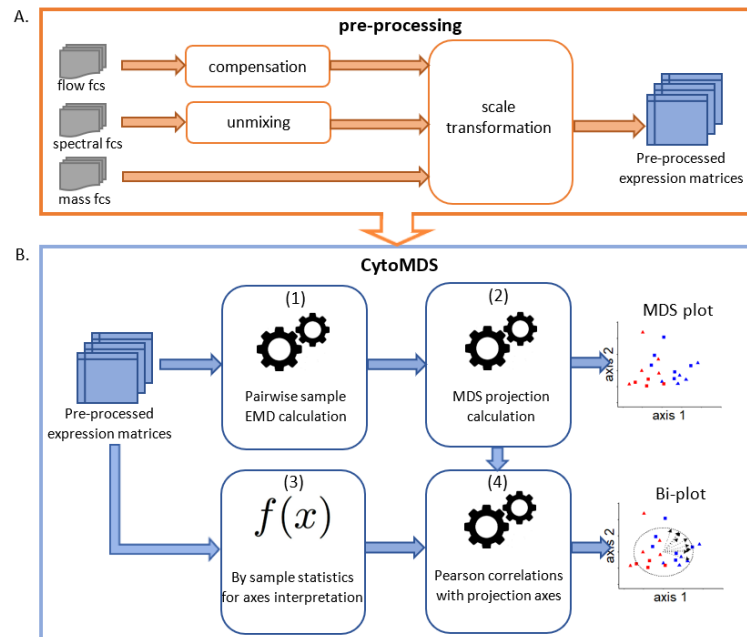


Figure 1. Overview of the *CytoMDS* workflow. Pre-processing of the raw *fcs* files [A] should be done according to the workflow in the orange box, which includes scale transformation, and might include compensation or unmixing depending on the type of data. The *CytoMDS* method itself [B] takes as input one pre-processed expression matrix per sample. First, *Earth Mover's Distances (EMD)* are calculated for each pair of samples (1). Second, the resulting distance matrix is projected using *Multi Dimensional Scaling (MDS)* (2), producing a *MDS* plot. Optionally, in order to interpret *MDS* coordinates and plot directions, *CytoMDS* can compute a set of chosen sample statistics (3). *CytoMDS* then calculates Pearson correlations of these statistics with the *MDS* coordinates (4), producing one or several bi-plot(s).

Distance calculation

In order to characterize the dissimilarity between samples, the *Earth Mover's Distance (EMD)*, a.k.a. the *Wasserstein distance of order one*, is chosen as our approach. Considering two samples a and b , which are represented by their respective M -dimensional distribution of expression intensities for the M markers, the *EMD* can be seen as the minimal cost (or effort) needed for transporting the distribution mass of one sample to the other, where the cost of transporting one unit of mass from location x to location y is given by the Euclidean distance between x and y (Rubner, Tomasi, and Guibas, 2000).

The *EMD* has been shown to be suitable for quantifying dissimilarities between cytometry samples, and to provide better insight than other types of metrics for this purpose (Orlova et al., 2016). It has been used in different applications of cytometry data analysis, especially in the evaluation of batch correction methods (Van Gassen et al., 2020, Pedersen et al., 2022) or for obtaining a sample representation in relation with different phenotypes (Yi and Stanley, 2022).

Here, for computational efficiency given the potentially high number of markers, we approximate the multi-dimensional *EMD*, by the sum of the M 1D *EMD* for each marker $m : 1, \dots, M$. Assuming

that each unidimensional marker expression distribution has been discretized in 1D histograms consisting of a number of bins with specific location and weight, the 1D *EMD* is then formally defined as follows:

- Let a_i and w_i^a ($i : \{1, \dots, r\}$) be respectively the location and weight assigned to bin i of the 1D histogram of marker m in sample a .
- Let b_j and w_j^b ($j : \{1, \dots, s\}$) be respectively the location and weight assigned to bin j of the 1D histogram of marker m in sample b .

The empirical distribution functions $F(t)$ and $G(t)$ of the marker m intensity, respectively for sample a and b , are therefore: $F(t) = \sum_{i=1}^r w_i^a 1\{a_i \leq t\}$ and $G(t) = \sum_{j=1}^s w_j^b 1\{b_j \leq t\}$.

The 1D *EMD* between marker m intensities of samples a and b is then:

$$W_1(F, G) = \int_{-\infty}^{\infty} |F(z) - G(z)| dz \quad (1)$$

To compute the *EMD* for each marker (Equation 1), we take advantage of the R package *transport* (Schuhmacher et al., 2024). This computation is then repeated for each pair of samples, leading to a symmetric pairwise distance matrix having N rows and N columns - with N the number of samples. Its diagonal is filled with zeros.

Projection algorithm

The second step consists of projecting the distance matrix of dimensions $N \times N$ onto a low dimensional embedding with p dimensions. As our goal is to visualize sample distances to identify outliers or batch effects, we chose a method from the *Multi Dimensional Scaling (MDS)* family, since they are good at preserving distances and the global data structure (Lee and Verleysen, 2007), in contrast to popular local neighbourhood preserving methods such as *t-SNE* (van der Maaten and Hinton, 2008) and *UMAP* (McInnes, Healy, and Melville, 2018).

Furthermore, because the *EMD* is a metric but is not euclidean, we need to resort, for the projection, to *non classical metric MDS*. It is a non linear method minimizing a specific criterion called *stress* (Lee and Verleysen, 2007) that represent a quadratic error of projection and is defined as follows:

$$\sigma(X) = \sum_{i < j} (\delta_{ij} - d_{ij}(X))^2 \quad (2)$$

where X is the low-dimensional embedding, δ_{ij} is the original *EMD* between sample i and sample j , and d_{ij} is the euclidean distance between the corresponding projections in X .

In order to minimize the stress criterion (Equation 2), *CytoMDS* uses the *SMACOF* (Scaling by MAjorizing a COmplicated Function) algorithm, implemented in the *smacof* R package (de Leeuw and Mair, 2009). Finally, the projection axes of the p -dimensional embedding are linearly transformed using *PCA*. This last step results in new projection coordinates that are ordered in decreasing percentages of variance, as in a classical *PCA* visualization. However, here these percentages are with respect to the total variance associated with the embedding, which differs from the total variability of the data.

Quality of projection

CytoMDS provides different diagnostic tools to assess the quality of the low dimensional projection. The first one is a visualization plot, called *Shepard's diagram*, which allows to assess how close the projected euclidean distances are to their *EMD*, for each sample pair. Examples of such diagrams can be found in Figure S1 and Figure S3. The second diagnostic tool is a global indicator of the projection quality, the *pseudo R²*, which assesses to which extent the pairwise sample *EMD* can be explained by the low dimensional euclidean distances on the projection. It is defined as follows:

$$pseudoR^2 = 1 - \frac{\sum_{i < j} (\delta_{ij} - d_{ij}(X))^2}{\sum_{i < j} (\delta_{ij} - \bar{\delta})^2}, \text{ where } \bar{\delta} \text{ is the overall mean of pairwise } EMD. \quad (3)$$

This $pseudoR^2$ plays a central role in *CytoMDS* as, by default, p , the number of dimensions of the low dimensional embedding, is tuned automatically to reach a specific $pseudoR^2$ target for the projection, 0.95 by default. Finally, additional indicators from the *SMACOF* algorithm can also be recovered, such as the minimised value of the stress criterion (Equation 2) and the contribution of each sample to the stress, i.e. the *stress per point* (de Leeuw and Mair, 2009).

Coordinates and directions interpretation using bi-plots

Bi-plots are plots that are overlaid to the *MDS* projection, allowing to interpret directions of interest by relating them to well-chosen sample characteristics (Borg, Groenen, and Mair, 2018). Creating a bi-plot involves choosing *external scales*, i.e. sample specific characteristics that can be fitted to the *MDS* space. Possible choices are summary statistics of marker intensities, e.g. univariate mean, standard deviation, median or other quantiles, etc., or any other sample representative characteristics (multivariate statistics, number of events in sample, etc.). *CytoMDS* calculates the Pearson correlations of the *external scales* with respect to the visualized *MDS* coordinates, and overlays arrows on the plot, one for each *external scale*, having coordinates equal to their respective correlations. A unit 1 correlation circle is also plotted as landmark for visual convenience (e.g. in Figure 2, [C] and [D]).

Example datasets

In order to demonstrate *CytoMDS* in subsequent sections, we make use of three datasets.

HBV Chronic Mouse dataset

The *HBV Chronic Mouse* dataset is a flow cytometry dataset, collected in a preclinical study aimed at assessing the effect of different therapeutic vaccine regimes on the immune response of Hepatitis B Virus transduced mice. This dataset has been described in Hauchamps et al., 2024. In short, it contains 55 flow cytometry mouse liver lymphocyte samples, analyzed with a 12-channel flow cytometry panel. The 55 mice were split in 5 experimental groups corresponding to different vaccine regimens, and the flow cytometry data were acquired in two flow cytometry batches performed on two different days (D91 and D93). The *HBV Chronic Mouse* dataset is available on Zenodo (DOI:10.5281/zenodo.8425840).

ImmunoSenescence Human PBMC dataset

The *ImmunoSenescence Human PBMC* dataset is also a flow cytometry dataset. This dataset has been generated in the context of the development of a 26 markers panel that aimed at phenotyping all the main immune cell populations present in PBMC (Peripheral Blood Mononuclear Cell) samples. This readout development was part of a study that aimed at assessing the effect of age on the human immune system. The dataset contains 20 *fcs* files, corresponding to the flow cytometry data acquisition of 15 PBMC samples. These were collected and cryopreserved from healthy individuals, split into two age groups - younger and older adults - and were part of two different GSK biobanks. There is therefore a potential confounding effect between the biological signal of interest - linked to ageing - and the two different sample origins. The flow cytometry data acquisition itself was performed in two different acquisition rounds (a *former* and a *later* data acquisition), and 5 of the biological samples were acquired in both rounds (see Table S1). Sampling and research were approved by the Belgian Ethics committee and written informed consent was obtained from all participants prior to sample collection, in accordance with the Helsinki Declaration.

Krieg_Anti_PD1 dataset

The *Krieg_Anti_PD1* dataset is a mass cytometry dataset (Krieg et al., 2018), consisting of 20 baseline samples (prior to treatment) of peripheral blood from melanoma skin cancer patients subsequently treated with anti-PD-1 immunotherapy. The samples were split across 2 conditions (non-responders and responders) and in 2 acquisition batches. The goal of the study was to identify

biomarkers of responsiveness to immunotherapy at baseline. This dataset is available in FlowRepository (ID: FR-FCM-ZYL8) and is directly accessible in R from the Bioconductor data package *HDCytoData* (Weber and Soneson, 2019).

Results

Identifying low quality samples

In this section, we demonstrate the use of *CytoMDS* on the *HBV Chronic Mouse* dataset, with QC as a specific goal. The raw data were pre-processed by applying compensation and scale transformation using respectively bi-exponential transformations (Parks, Roederer, and Moore, 2006) for fluorescent channels, and linear transformations for *FSC* and *SSC* channels. Coefficients of the linear transformations were calibrated as to align 5 and 95 quantiles of *FSC/SSC* channels to those of a reference fluorescent channel. *CytoMDS* was used to compute *EMD* between each pair of samples. Here, since the focus is on tracking sample quality issues, we limited ourselves to four specific channels as input of the *EMD* calculation, i.e. *FSC-A*, *FSC-H*, *SSC-A* and the *Live/Dead* channels, which are the channels used for pre-gating (den Braanker, Bongenaar, and Lubberts, 2021). Finally the low dimensional projection was computed for the 55 samples of the dataset.

The *Shepard's diagram*, in Figure S1, illustrates the projection quality. It shows that most pairwise *EMD* are close to the 45 degree identity line. Also, the *pseudo R²* is 0.9679, hence above the 0.95 threshold (chosen by default), with only two projected dimensions.

The projection, in Figure 2 [A], highlights two major clusters of samples. On the positive side of the x coordinate, we can find most of the samples acquired on day D93 (blue squares). The other cluster of samples, on the negative side of the x coordinate, includes mostly the samples from day D91 (red triangles), but also a minority of day D93 samples. Therefore, these clusters cannot be attributed solely to a batch effect of acquisition time point. Two bi-plots using medians and standard deviations (Figure 2, respectively [C] and [D]) show that the x coordinate is strongly correlated, in a negative manner, with both the *FSC-A* channel median and *FSC-A* standard deviation, and positively correlated with *SSC-A* standard deviation. *FSC-A* and *SSC-A* are therefore taken as plot axes in Figure 2 [B], showing 2D plots for two representative samples, selected for their extreme opposite position on the first axis of the projection. Comparing these two plots reveals that sample *D93_A05* is a low quality sample where the cell population of interest - liver lymphocytes - is extremely small, and where most events seem to correspond to debris or dead/dying cells, whereas sample *D93_G03* is a good quality sample. Actually, as shown in Figure S2, the same pattern can be seen in all samples belonging to the low quality cluster.

As a conclusion, the *CytoMDS* projection of the *HBV Chronic Mouse* dataset, complemented by bi-plots, has allowed to identify a sample cluster corresponding to low quality samples, in terms of relative percentage of events belonging to the cell population of interest.

Highlighting batch effects

In this section, we apply *CytoMDS* on the *ImmunoSenescence Human PBMC* dataset. As in the previous section, we pre-processed the data by applying compensation - based on a manually adjusted compensation matrix - and scale transformed the channel intensities - using bi-exponential transformations or linear transformations as in previous section. The pairwise sample *EMD* were calculated using the *FSC-A*, *FSC-H*, *SSC-A* and *Live/Dead* channels. In contrast to the *HBV Chronic Mouse* dataset, *CytoMDS* required 3 dimensions to reach the *pseudo R²* quality threshold of 0.95. The *Shepard's diagram* is shown on Figure S3.

Figure 3 shows the low dimensional projection according to 2 different combinations of *MDS* axes. Figure 3 [A] shows the projection on axes 1 and 2, which reveals a clear separation between the samples of the former (triangles), and later (squares) data acquisitions. This is an obvious batch

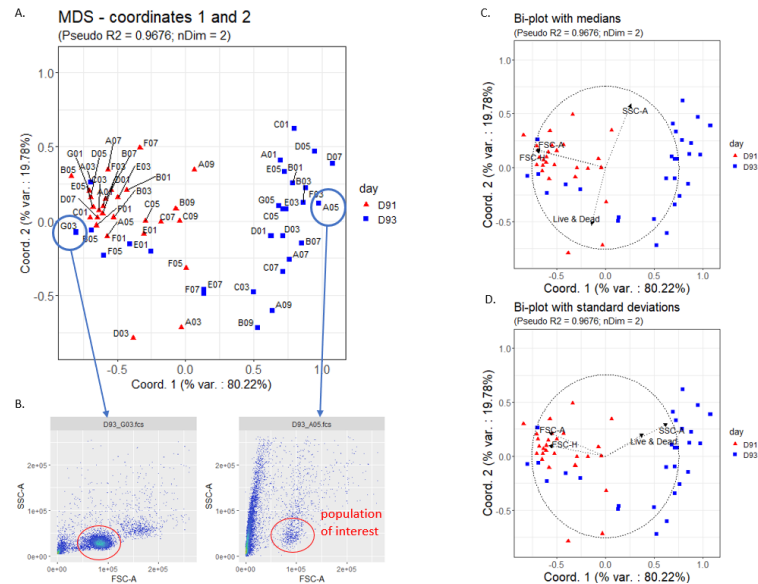


Figure 2. The low dimensional projection of the *HBV Chronic Mouse* dataset [A] reveals a clear split between two sample groups, not fully attributable to a batch effect linked to day of acquisition - since a minority of day D93 samples belong to the left sample group. Bi-plots overlaying respectively channels medians [C], and standard deviations [D] show that the x axis, which is the direction discriminating the two sample groups, is strongly negatively correlated both with FSC channel median and FSC standard deviation, and also positively correlated with SSC standard deviation. A comparison of the FSC vs. SSC 2D representation of selected samples having opposite x coordinates [B] shows that the right sample group contains low quality samples, with very few events belonging to the cell population of interest.

effect. Using bi-plots to interpret further the direction of separation highlights that this batch effect is almost exclusively explained by a difference of location and scale of the *Live/Dead* channel (Figure S4). This effect is most probably caused by technical differences between the two data acquisitions for this particular channel. These could be, for example, due to different cytometer set-ups, or different staining efficiencies for the different batches.

On Figure 3 [B], samples of the 2 experimental groups (old vs. young donors) can be separated along a direction that is a linear combination of axes 2 and 3. Since we have only included pre-processing channels in the distance calculation, this separation is unlikely to reflect the biological signal of interest. In fact, it probably highlights another batch effect, which is confounded with experimental groups. We show this by interpretation of the direction of separation, which is here less obvious. Bi-plots using medians (Figure S5 [C]), or using standard deviations (not shown), are not able to identify any strong correlation with the direction of separation. However, Figure S5 [D] shows that the latter is strongly correlated with 10th quantiles of FSC and SSC channels, highlighting the tendency for young adult samples to contain more debris than old adult samples (Figure S5 [B]). This is most probably due to differences in sample handling between these two groups of samples, which were taken from different biobanks (see Methods). Finally, one can also notice that, apart from the first coordinate, driven by *Live/Dead* channel intensities, sample data points with the same label number tend to colocate (ellipses on Figure 3 [B]). This reflects the fact that

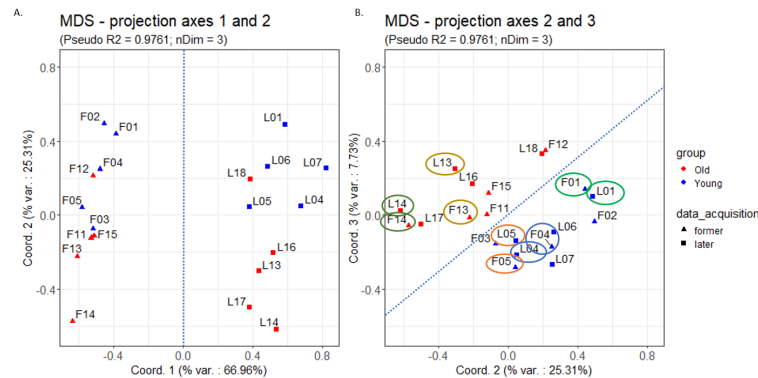


Figure 3. Low dimensional projection of the *ImmunoSenescence Human PBMC* dataset. (A) The projection on axes 1 and 2 reveals a clear batch effect between data points belonging to the former (triangles) and the later data acquisitions (squares). (B) The two groups ('old' in red vs. 'young' in blue), can be separated along a direction that is a linear combination of axes 2 and 3. On the same plot, one can also notice the proximity of the data points corresponding to the biological samples from the same donor (ellipses of different colours: F01 vs. L01, F04 vs. L04, F05 vs. L05, F13 vs. L13, F14 vs. L14).

these pairs correspond to two acquisitions of the same original biological sample.

To summarize this section, the *CytoMDS* projection of the *ImmunoSenescence Human PBMC* dataset, according to two different combinations of axes, allowed to highlight separations of sample groups that could be attributed to two different batch effects, respectively linked to data acquisition time point and sample origin. Besides, proximity of the samples corresponding to the same biological donor could be shown by the projection.

Highlighting biological signal between sample groups

CytoMDS projections can also reveal biological signal between different groups of samples, provided that this signal is sufficiently strong. In this section, we apply *CytoMDS* on the *Krieg_Anti_PD1* dataset. As this dataset is a mass cytometry dataset, pre-processing involves applying an *arcsinh()* scale transformation, but no channel compensation. Here, we used all 25 channels as input for the *EMD* computation, and projected the pairwise distances between the 20 samples using *CytoMDS* with 2 dimensions (Figure 4, [A]).

The *Krieg_Anti_PD1* dataset contains a strong batch effect, due to sample acquisition on two different days (Krieg et al., 2018). This batch effect is clearly visible on Figure 4, [A]: *batch29* samples (squares) tend to be located on the left of the plot, while the *batch23* samples (triangles) tend to be located on the right. Therefore, we performed a batch correction to this dataset, by applying the *ComBat* method (Johnson, Li, and Rabinovic, 2007), implemented in the *cyCombine* package (Pedersen et al., 2022), and using the responder status - on anti-PD-1 immunotherapy - as covariate. Figure 4 [B] displays the projection after batch correction, showing a more sensible separation of the two biological condition groups - responders (blue) vs. non responders (red) - and a better mixing of samples from the two batches.

Figure S6 shows bi-plots with medians, for the projection on axes 1 and 2, after batch correction. In the original study (Krieg et al., 2018), it was highlighted that the frequency of a small subpopulation of monocytes in baseline samples was positively associated with responder status following immunotherapy treatment. The left bi-plot of Figure S6 shows that the 11 markers involved in the

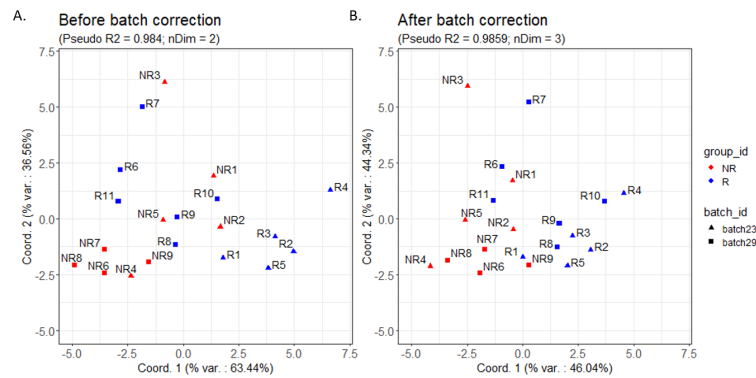


Figure 4. Low dimensional projection of the *Krieg_Anti_PD1* dataset, before batch correction [A], and after batch correction [B], on the first two axes. [A] The projection reveals a clear batch effect between *batch29* samples (squares), which tend to be located on the left side of the plot, and the *batch23* samples (triangles), which tend to be located on the right side. [B] The batch correction results in a better mix between samples of different batches. As a consequence, the separation between the two condition groups - responders (blue) vs. non responders (red) - is more sensible.

definition of this rare population tend to be positively correlated with the direction of separation between responders vs non responders, while this is not the case on the right bi-plot, which shows the other 14 markers.

As a conclusion of this section, we showed that *CytoMDS* can be used, as an exploratory data visualization tool, to highlight biological signal between groups of samples, provided this signal is sufficiently strong.

Discussion

CytoMDS enables fast and global detection of data quality issues

In the Results section, we have provided three examples of using *CytoMDS* to detect sample quality issues. For the *HBV Chronic Mouse* dataset, the issue was specific to a group of samples for which an improper sample handling problem had shrunk the population of interest to a small percentage of the acquired events. For the *ImmunoSenescence Human PBMC* dataset and the *Krieg_Anti_PD1* dataset, the issues were the presence of one or several batch effects, sometimes confounded with the biological condition of interest. Beyond the identification of these quality issues, the ability to link these to specific sample characteristics, hence to potential root causes, is essential to the users, who will more easily make educated decisions about the encountered problems. For this purpose, the *CytoMDS* method has several strengths:

- It generates a global view of all samples, where the distances between points can be quantitatively interpreted.
- It provides bi-plots, a powerful interpretation tool which enables to correlate plot coordinates to sample characteristics. As a benefit, valuable insight on the potential cause(s) of the observed pattern(s) can be inferred.
- It allows unexpected batch effects to be revealed, in addition to those suspected. Indeed, it is not necessary to know in advance which samples belong to which batch. The *MDS* projection itself can be used to uncover patterns, which can in turn be dissected with the help of bi-plots, in order to highlight the specific sample characteristics that explain the observed groupings

of samples. Similarly, this process can be used to detect outlier samples.

These strengths are not found in tSNE or UMAP, which are other popular techniques used to highlight batch effects, and which typically project the events of all samples of a dataset at once with low dimensions. An example of a 2 dimensional tSNE plot of the *HBV Chronic Mouse* dataset is provided in Figure S7. This plot suggests a batch effect driven by acquisition day, while we know from Figure 2 that this effect is driven by a quality issue observed for only a subset of the samples acquired on a specific day. However, as is shown in Figure S7, due to the high number of samples and high number of cells, there is no easy way to precisely identify the impacted samples, nor to link the quality issue to specific sample characteristics. In contrast, the *CytoMDS* projection (Figure 2) stays informative when the number of samples increases to tens or even a few hundreds.

Nowicka et al., 2017 proposed another *MDS* projection method for QC of cytometry data. This method, however, is not based on distributional distances between samples, but uses the medians of each channel to summarize the information contained in each sample. As a result, this method can lead to projections that are very similar to the *CytoMDS* ones, only when sample distances are driven mostly by the channel medians, but is insensitive to other differences between sample distributions. Figure S8 shows a comparative example, with low dimensional projections of the *ImmunoSenescence Human PBMC* dataset. *CytoMDS* allows to identify the distributional characteristics that drive the dissimilarity between the samples at hand, which are not necessarily the channel medians.

***CytoMDS* can be used at different stages of the data analysis pipeline**

In Figure 1, we proposed a workflow where only compensation and scale transformation were performed prior to *CytoMDS* projection. However, as a data exploration tool, *CytoMDS* can be run at any stage of the data analysis pipeline. In the Results section, the *CytoMDS* projections were compared before and after batch correction. In general, the iterative use of *CytoMDS* can uncover quality issues at any stage of the data analysis pipeline.

Computation time and scalability to larger datasets

In the Supplementary Results, we present a benchmarking of the *CytoMDS EMD* computation time. As is shown in Figures S9 and S10, the number of markers impacts the *EMD* computation time linearly, while sub-sampling allows to mitigate the effect of a large number of events per sample, with no material impact on precision. As a result, a single *EMD* calculation requires less than one second.

However, for datasets with hundreds of cytometry samples, computing the *EMD* between all sample pairs is a heavy computational task. A first pitfall might arise when loading the whole dataset in memory, which might not be possible due to its size. Another challenge is that calculating a matrix of pairwise distances has a computational complexity of $O(N^2)$, which can lead to long computation time for large datasets. The *CytoMDS* package provides two mechanisms to mitigate these issues. First, to handle datasets of greater size than the available computer RAM, *CytoMDS* provides a distance calculation method that dynamically loads data on the fly when calculating a pairwise sample distance, and releases the corresponding memory afterwards. Second, the user can instruct *CytoMDS* to parallelize the distance calculation process. In that case, the calculation engine takes advantage of the *BiocParallel* package (Morgan et al., 2023) to dispatch the calculation on several workers. The calculation engine then automatically creates balanced worker tasks, corresponding to blocks of the distance matrix, minimizing the overlap between data loaded in memory by different workers. These mechanisms have allowed to decrease the computation time of the full pairwise distance matrix of a dataset of 1400 mass cytometry samples with 43 parameters, from about 7 days to about 1.3 hours on a 14 cores user laptop (Granjeaud et al., 2024).

Positioning, limitation and possible future extension

CytoMDS has been developed as an exploratory visualization method, aiming at facilitating the quality control process, for outliers and batch effects detection. As was shown in the *Krieg Anti_PD1* example, this exploratory visualization can also highlight sample clusters that correspond to different conditions of interest, i.e. pointing to the existence of a biological signal. However, identification of such signal requires a dedicated differential expression/abundance analysis method (e.g. Weber et al., 2019, Seiler et al., 2021). In particular, a biological signal of interest consisting in the existence (or absence) of a rare immune cell population can translate into very subtle sample distribution differences involving conditional distributions of multiple combined marker intensities. These cannot be captured by *EMD* calculated as sum of marginal distributions of markers, as is done in *CytoMDS* (see Methods). A possible extension of this work could be to refine the calculation of pairwise sample *EMDs*, to incorporate the information on joint marker distributions. This would however involve a difficult computational challenge, especially in high dimensional cytometry datasets. Finally, the current implementation has already shown, throughout the different real dataset examples, its ability to highlight various data quality issues, and link them to potential root causes, which is the main goal of our method.

Declarations

Funding

This work was funded by GlaxoSmithKline Biologicals S.A., under a cooperative research and development agreement between GlaxoSmithKline Biologicals S.A. and de Duve Institute (UCLouvain).

Competing Interests

- S.D., S.T., and D.L. are employees of the GSK group of companies and report ownership of GSK shares.
- S.T. is listed as inventor on patents owned by the GSK group of companies.
- P.H. is a student at the de Duve Institute (UCLouvain) and participates in a post graduate studentship program at GSK.
- L.G. reports no competing interest.

Author contributions

P.H.: conceptualization (lead); methodology; software (lead); writing - original draft (lead). S.D.: methodology (supporting); writing - original draft (supporting). S.T.: supervision (supporting); writing - review & editing (supporting). D.L.: supervision; methodology (supporting); writing - review & editing. L.G.: supervision; conceptualization (supporting); methodology; software (supporting); writing - review & editing.

Availability of data and materials

Raw cytometry data files are available either on Zenodo (DOI:10.5281/zenodo.10572228) or FlowRepository (ID: FR-FCM-ZYL8). The *ImmunoSenescence Human PBMC* dataset is available upon request.

Code availability

CytoMDS package is available on Bioconductor: <https://www.bioconductor.org/packages/release/bioc/html/CytoMDS.html>

All code needed to reproduce the results presented in the current article is available on the following GitHub repository: <https://github.com/UCLouvain-CBIO/2024-CytoMDS-code>.

Acknowledgments

The authors would like to thank Mehdi Hamrouni and Babak Bayat (GSK) for the provision of the *HBV Chronic Mouse* dataset, as well as Sébastien Baudart and Michaël Ska (GSK) for the data acqui-

sition of the *ImmunoSenescence Human PBMC* dataset.

This preprint was created using the LaPreprint template (<https://github.com/roaldarbol/lapreprint>) by Mikkel Roald-Arbøl .

References

- Borg, Ingwer, Patrick J.F. Groenen, and Patrick Mair (2018). *Applied Multidimensional Scaling and Unfolding*. Springer International Publishing. ISBN: 9783319734712. DOI: [10.1007/978-3-319-73471-2](https://doi.org/10.1007/978-3-319-73471-2). URL: <http://dx.doi.org/10.1007/978-3-319-73471-2>.
- Chen, Xiaoying, Bo Zhang, Ting Wang, Azad Bonni, and Guoyan Zhao (2020). "Robust principal component analysis for accurate outlier sample detection in RNA-Seq data". In: *BMC Bioinformatics* 21.1, p. 269.
- de Leeuw, Jan and Patrick Mair (2009). "Multidimensional Scaling Using Majorization: SMACOF in R". In: *J. Stat. Softw.* 31, pp. 1–30.
- den Braanker, Hannah, Margot Bongenaar, and Erik Lubberts (2021). "How to Prepare Spectral Flow Cytometry Datasets for High Dimensional Data Analysis: A Practical Workflow". In: *Front. Immunol.* 12, p. 768113.
- Goh, Wilson Wen Bin, Wei Wang, and Limsoon Wong (2017). "Why Batch Effects Matter in Omics Data, and How to Avoid Them". In: *Trends Biotechnol.* 35.6, pp. 498–507.
- Goh, Wilson Wen Bin and Limsoon Wong (2017). "Protein complex-based analysis is resistant to the obfuscating consequences of batch effects — a case study in clinical proteomics". In: *BMC Genomics* 18.Suppl 2, p. 142.
- Granjeaud, Samuel, Naoill Abdellaoui, Anne-Sophie Chretien, Eloise Woitrain, Laurent Pineau, Sandro Ninni, Alexandre Harari, Marion Arnaud, David Montaigne, Bart Staels, David Dombrowicz, and Olivier Molendi-Coste (2024). "CytoBatchNorm: an R package with graphical interface for batch effects correction of cytometry data".
- Hammill, Dillon (2021). *CytoExploreR: Interactive Analysis of Cytometry Data*. R package version 1.1.0. URL: <https://github.com/DillonHammill/CytoExploreR>.
- Hartmann, Felix J, Joel Babbod, Pier Federico Gherardini, El-Ad D Amir, Kyle Jones, Bitá Sahaf, Diana M Marquez, Peter Krutzik, Erika O'Donnell, Natalia Sigal, Holden T Maecker, Everett Meyer, Matthew H Spitzer, and Sean C Bendall (2019). "Comprehensive Immune Monitoring of Clinical Trials to Advance Human Immunotherapy". In: *Cell Rep.* 28.3, 819–831.e4.
- Hauchamps, Philippe, Babak Bayat, Simon Delandre, Mehdi Hamrouni, Marie Toussaint, Stephane Temmerman, Dan Lin, and Laurent Gatto (2024). "CytoPipeline and CytoPipelineGUI: a Bioconductor R package suite for building and visualizing automated pre-processing pipelines for flow cytometry data". In: *BMC Bioinformatics* 25.1, p. 80.
- Hauchamps, Philippe and Laurent Gatto (2024). *CytoMDS: Low Dimensions projection of cytometry samples*. R package version 1.0.0. URL: <https://uclouvain-cbio.github.io/CytoMDS>.
- Huber, Wolfgang, Vincent J Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S Carvalho, Hector Corrada Bravo, Sean Davis, Laurent Gatto, Thomas Girke, Raphael Gottardo, Florian Hahne, Kasper D Hansen, Rafael A Irizarry, Michael Lawrence, Michael I Love, James MacDonald, Valerie Obenchain, Andrzej K Oleś, Hervé Pagès, Alejandro Reyes, Paul Shannon, Gordon K Smyth, Dan Tenenbaum, Levi Waldron, and Martin Morgan (2015). "Orchestrating high-throughput genomic analysis with Bioconductor". In: *Nat. Methods* 12.2, pp. 115–121.
- Johnson, W Evan, Cheng Li, and Ariel Rabinovic (2007). "Adjusting batch effects in microarray expression data using empirical Bayes methods". In: *Biostatistics* 8.1, pp. 118–127.
- Kare, Aris J, Lisa Nichols, Ricardo Zermeno, Marina N Raie, Spencer K Tumbale, and Katherine W Ferrara (2023). "OMIP-095: 40-Color spectral flow cytometry delineates all major leukocyte populations in murine lymphoid tissues". In: *Cytometry A* 103.11, pp. 839–850.
- Krieg, Carsten, Malgorzata Nowicka, Silvia Guglietta, Sabrina Schindler, Felix J Hartmann, Lukas M Weber, Reinhard Dummer, Mark D Robinson, Mitchell P Levesque, and Burkhard Becher (2018).

- "High-dimensional single-cell analysis predicts response to anti-PD-1 immunotherapy". In: *Nat. Med.* 24.2, pp. 144–153.
- Lee, John A and Michel Verleysen (2007). *Nonlinear Dimensionality Reduction*. Springer New York. ISBN: 9780387393506. DOI: [10.1007/978-0-387-39351-3](https://doi.org/10.1007/978-0-387-39351-3). URL: <http://dx.doi.org/10.1007/978-0-387-39351-3>.
- Liechti, Thomas, Lukas M Weber, Thomas M Ashhurst, Natalie Stanley, Martin Prlic, Sofie Van Gassen, and Florian Mair (2021). "An updated guide for the perplexed: cytometry in the high-dimensional era". In: *Nat. Immunol.* 22.10, pp. 1190–1197.
- McInnes, Leland, John Healy, and James Melville (2018). "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". In: arXiv: [1802.03426](https://arxiv.org/abs/1802.03426) [stat.ML].
- Morgan, Martin, Jiefei Wang, Valerie Obenchain, Michel Lang, Ryan Thompson, and Nitesh Turaga (2023). *BiocParallel: Bioconductor facilities for parallel evaluation*. R package version 1.37.0. DOI: [10.18129/B9.bioc.BiocParallel](https://doi.org/10.18129/B9.bioc.BiocParallel). URL: <https://bioconductor.org/packages/BiocParallel>.
- Newell, Evan W and Yang Cheng (2016). "Mass cytometry: blessed with the curse of dimensionality". In: *Nat. Immunol.* 17.8, pp. 890–895.
- Nowicka, Malgorzata, Carsten Krieg, Helena L Crowell, Lukas M Weber, Felix J Hartmann, Silvia Guglietta, Burkhard Becher, Mitchell P Levesque, and Mark D Robinson (2017). "CyTOF workflow: differential discovery in high-throughput high-dimensional cytometry datasets". In: *F1000Res*. 6, p. 748.
- Orlova, Darya Y, Noah Zimmerman, Stephen Meehan, Connor Meehan, Jeffrey Waters, Eliver E B Ghosn, Alexander Filatenkov, Gleb A Kolyagin, Yael Gernez, Shanel Tsuda, Wayne Moore, Richard B Moss, Leonore A Herzenberg, and Guenther Walther (2016). "Earth Mover's Distance (EMD): A True Metric for Comparing Biomarker Expression Levels in Cell Populations". In: *PLOS ONE* 11.3, e0151859.
- Parks, David R, Mario Roederer, and Wayne A Moore (2006). "A new "Logicle" display method avoids deceptive effects of logarithmic scaling for low signals and compensated data". In: *Cytometry A* 69.6, pp. 541–551.
- Pearson, Karl (1901). "On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11, pp. 559–572. DOI: [10.1080/14786440109462720](https://doi.org/10.1080/14786440109462720).
- Pedersen, Christina Bligaard, Søren Helweg Dam, Mike Bogetoft, Barnkob, Michael D Leipold, Noelia Purroy, Laura Z Rassenti, Thomas J Kipps, Jennifer Nguyen, James Arthur Lederer, Satyen Harish Gohil, Catherine J Wu, and Lars Rønn Olsen (2022). "cyCombine allows for robust integration of single-cell cytometry datasets within and across technologies". In: *Nat. Commun.* 13.1, p. 1698.
- Rubner, Yossi, Carlo Tomasi, and Leonidas J Guibas (2000). "The Earth Mover's Distance as a Metric for Image Retrieval". In: *Int. J. Comput. Vis.* 40.2, pp. 99–121.
- Saeys, Yvan, Sofie Van Gassen, and Bart N Lambrecht (2016). "Computational flow cytometry: helping to make sense of high-dimensional immunology data". In: *Nat. Rev. Immunol.* 16.7, pp. 449–462.
- Schmit, Taylor, Mitchell Klomp, and M Nadeem Khan (2021). "The Application of Flow Cytometry for Simultaneous and Multi-parametric Analysis of Heterogenous Cell Populations in Basic and Clinical Research". In: *Methods Mol. Biol.* 2223, pp. 183–200.
- Schuhmacher, Dominic, Björn Bähre, Carsten Gottschlich, Valentin Hartmann, Florian Heinemann, and Bernhard Schmitzer (2024). *transport: Computation of Optimal Transport Plans and Wasserstein Distances*. R package version 0.14-7. URL: <https://cran.r-project.org/package=transport>.
- Seiler, Christof, Anne-Maud Ferreira, Lisa M Kronstad, Laura J Simpson, Mathieu Le Gars, Elena Vendrame, Catherine A Blish, and Susan Holmes (2021). "CytoGLMM: conditional differential analysis for flow and mass cytometry experiments". In: *BMC Bioinformatics* 22.1, p. 137.
- Spitzer, Matthew H and Garry P Nolan (2016). "Mass Cytometry: Single Cells, Many Features". In: *Cell* 165.4, pp. 780–791.

- Staats, Janet, Anagha Divekar, J Philip McCoy Jr, and Holden T Maecker (2019). "Guidelines for Gating Flow Cytometry Data for Immunological Assays". In: *Methods Mol. Biol.* 2032, pp. 81–104.
- van der Maaten, Laurens and Geoffrey Hinton (2008). "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86, pp. 2579–2605. URL: <http://jmlr.org/papers/v9/vandermaaten08a.html>.
- Van Gassen, Sofie, Brice Gaudilliere, Martin S Angst, Yvan Saeys, and Nima Aghaeepour (2020). "CytoNorm: A Normalization Algorithm for Cytometry Data". In: *Cytometry A* 97.3, pp. 268–278.
- Weber, Lukas M, Malgorzata Nowicka, Charlotte Soneson, and Mark D Robinson (2019). "diffcyt: Differential discovery in high-dimensional cytometry via high-resolution clustering". In: *Commun Biol* 2, p. 183.
- Weber, Lukas M and Charlotte Soneson (2019). "HDCytoData: Collection of high-dimensional cytometry benchmark datasets in Bioconductor object formats". In: *F1000Res*. 8, p. 1459.
- Yi, Haidong and Natalie Stanley (2022). "CytoEMD: detecting and visualizing between-sample variation in relation to phenotype with earth mover's distance". In: *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. BCB '22 Article 28. Northbrook, Illinois: Association for Computing Machinery, pp. 1–14.

Chapter 9

MDSvis, a Shiny application for interactive visualisation of Multi Dimensional Scaling projections

9.1 Context

While presenting our visual [QC](#) tool *CytoMDS* to potential users, we recognised that requiring R scripting was posing a significant barrier for many scientists. Additionally, navigating the various views and options in *CytoMDS* could be cumbersome without an interactive [GUI](#).

To enhance accessibility, we developed a companion R package, *MDSvis*, which offers an interactive Shiny application for exploring *CytoMDS* projections. Given that metric [MDS](#) is widely used beyond cytometry, we also designed *MDSvis* to support broader applications in any domain where low dimensional distance-based visualisation aids data exploration.

9.2 Article

Pre-print

Hauchamps, Philippe, Andrea Vicini, and Laurent Gatto. 2025. “MDSvis, a Shiny Application for Interactive Visualisation of Multi Dimensional Scaling Projections.” *bioRxiv*.

DOI: 10.1101/2025.06.23.660991.

MDSvis, a Shiny application for interactive visualisation of Multi Dimensional Scaling projections

Philippe Hauchamps ¹, Andrea Vicini ¹, Laurent Gatto ¹ 

¹Computational Biology and Bioinformatics, de Duve Institute, UCLouvain, Belgium

✉ **For correspondence:**
laurent.gatto@uclouvain.be

Present address: Computational Biology and Bioinformatics, de Duve Institute, UCLouvain, Av. Hippocrate 75, 1200 Brussels, Belgium

Data availability: The *Krieg_Anti_PD1* cytometry dataset is available from the *HDCytoData* Bioconductor R package. The *fsap* dataset is available from the *MDSvis* GitHub repository: github.com/UCLouvain-CBIO/MDSvis.

Software availability: The R package is available, under a GPL-3 license, from the *MDSvis* GitHub repository: github.com/UCLouvain-CBIO/MDSvis. The on-line documentation is available, under a CC-BY-4.0 license, from <https://uclouvain-cbio.github.io/MDSvis>.

Funding: P.H. is funded by GlaxoSmithKline Biologicals S.A., under a cooperative research and development agreement between GlaxoSmithKline Biologicals S.A. and de Duve Institute (UCLouvain).

Competing interests: P.H. is a student at the de Duve Institute (UCLouvain) and participates in a post graduate studentship program at GSK. A.V. and L.G. report no competing interest.

Abstract

Motivation

Visual exploratory data analysis is key for making sense of high dimensional data. In particular, Multi Dimensional Scaling (MDS), a low dimensional representation of data points, is frequently used in early stages of data analysis pipelines, for Quality Control (QC), batch effect detection and hypothesis generation. However, interpreting representations in more than two dimensions necessitates the mental integration of multiple two-dimensional (2D) plots - a process that greatly benefits from an interactive Graphical User Interface (GUI), where such plots can be easily and iteratively customised without the need for programming.

Results

Here, we present *MDSvis*, a Shiny application for user-interactive customised visualisation of MDS projections. *MDSvis* is based on the Bioconductor package *CytoMDS* for calculating MDS coordinates. We describe the functional workflow of the application, and illustrate two different use cases, which are detailed in the package vignettes. The first use case pertains to cytometry data analysis, which was the original motivation for building the application. The second use case demonstrates how to extend the use of *MDSvis* to MDS visualisation of any sample distance matrix.

Availability and implementation

MDSvis is implemented in the R programming language and is available in a GitHub repository: github.com/UCLouvain-CBIO/MDSvis, under a GPL-3 license.

Introduction

Visual exploratory data analysis is key for interpreting high dimensional data. In particular, low dimensional representations of data points, using e.g. Principal Component Analysis (PCA) [1], or Multi Dimensional Scaling (MDS) [2], are cornerstones of numerous data visualisation and analysis pipelines. Indeed, these multivariate visualisation methods can support early stages of the data analysis process such as Quality Control (QC), batch effect detection and hypothesis generation. In the field of computational flow cytometry [3], we have recently developed *CytoMDS*, a Bioconductor R package for low dimensional representation of single cell cytometry datasets [4]. *CytoMDS* performs fast calculation of Earth Mover's Distance (EMD) [5] between sample distributions, and generates a metric MDS [6] projection of the different samples, facilitating the detection of outliers

and batch effects.

Nevertheless, even in a low dimensional setting, the proper interpretation of representations in more than two dimensions necessitates the cognitive integration of multiple two-dimensional plots - a process that greatly benefits from an interactive Graphical User Interface (GUI), where such plots can be easily and iteratively customised without the need for programming. Hence, providing an interactive visualisation application on top of *CytoMDS*, for exploratory analysis of cytometry datasets, was the original motivation for the current work. However, since MDS has much wider use than cytometry data analysis, we designed the application such that it can also be useful for the MDS representation of any type of sample distances. Here, we present *MDSvis*, a Shiny application [7] enabling users to interactively explore and customise MDS projection plots.

Software implementation

Analytical workflow

An overview of the *MDSvis* analytical workflow is shown in Figure 1(A). It can be split into three consecutive steps. The first two steps are performed in an R script, while the last step is done interactively in the *MDSvis* Shiny application. The first step consists in obtaining a sample distance matrix, which can be done in two ways. In the first use case, corresponding to cytometry data analysis, a *CytoMDS* function is called to perform sample distance calculation, using pre-processed cytometry marker expression matrices as input. These matrices typically come from dedicated pre-processing pipelines where cytometry raw data files - in Flow Cytometry Standard (FCS) format - are read, scale transformed and, if applicable, compensated or unmixed. In the second use case, the user uploads a previously generated distance matrix directly. In a second step, in both use cases, *CytoMDS* is called to generate MDS projection results from the input distance matrix. Optionally, the user can also instruct *CytoMDS* to calculate sample statistics, which will be used in bi-plots, for plot interpretation [4]. These results are then saved as R Data Serialize (RDS) objects on disk. In the final step, the RDS objects are uploaded by the user into the *MDSvis* Shiny application, for interactive visualisation. The intermediate storage of the RDS objects allows for asynchronous execution of the visualisation step, hence it can be repeated without re-running the first two steps.

Application features

Figure 1(B) shows the different tabs of the Shiny application. In the initial 'Input files' tab (i), the user is prompted to import the RDS files containing the MDS projection results, sample statistics, and possibly additional sample characteristics - here called *phenoData* - that will be used to customise the plots. Then, the user can switch to the main 'View' tab (ii), where the MDS plot is displayed. The user can choose the x and y axes - among the available MDS dimensions. Besides, faceting variable, colour, shape and label of points can be customised by selecting corresponding sample data characteristics from drop-down lists. To facilitate user interpretation, a bi-plot can be overlaid onto the main plot, by selecting the sample statistics to be used - among the ones initially uploaded in the input tab. Finally, an additional interactive *plotly* plot [8] can be activated, which allows to zoom into parts of the plot and display additional data point information. In the 'General Settings' tab (iii), secondary plot layout settings, such as size of points, arrows and labels, can be modified by the user. All the generated plots can be exported as files in Portable Document Format (PDF), with customised title and plot dimensions (iv).

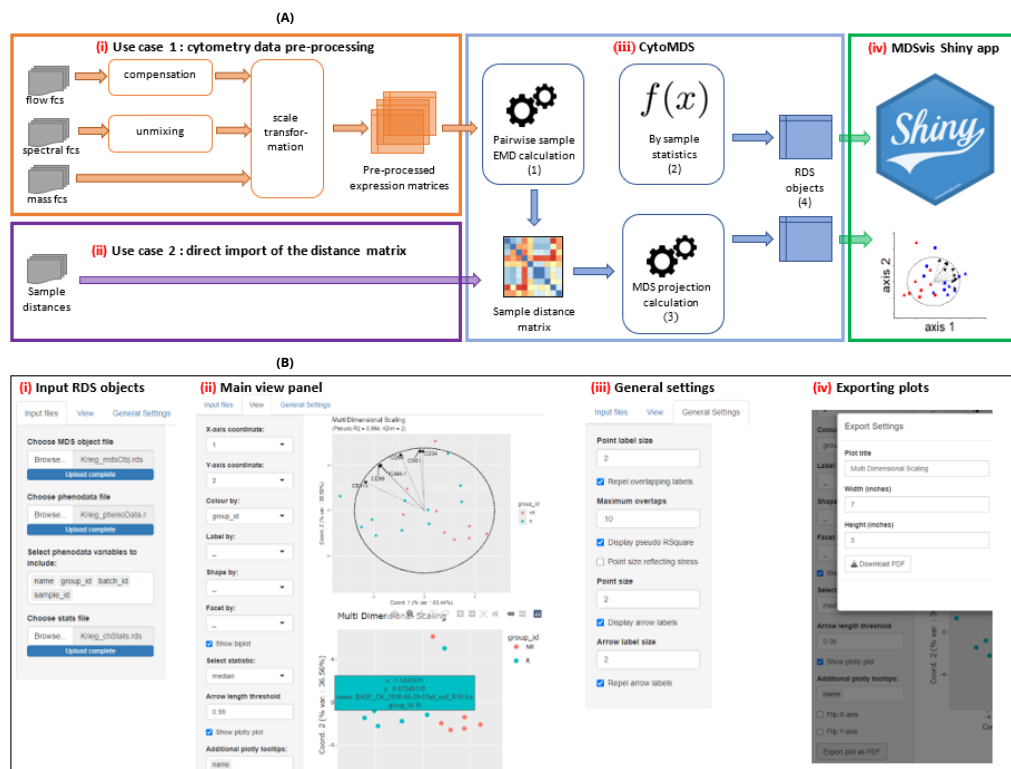


Figure 1. MDSvis overview. (A) Analytical workflow (figure adapted from [4]). (i) In use case 1, dedicated pre-processing is performed on cytometry data, and the pre-processed expression matrices are then used as input to *CytoMDS* sample distance calculation. (ii) In use case 2, the pairwise sample distance matrix is directly imported. (iii) *CytoMDS* calculation part: in use case 1, pairwise sample EMD distances are calculated from pre-processed cytometry expression matrices (1), and sample statistics can be automatically calculated (2). In both use cases, a MDS projection is calculated from the distance matrix (3). Calculation results are saved as RDS objects (4). (iv) RDS objects are input into the *MDSvis* Shiny application for interactive visualisation. (B) *MDSvis* layout and features. (i) Input tab, allowing RDS objects loading. (ii) Main view tab, displaying the MDS projection. Main plot attributes can be customised using sample data characteristics, and an additional *plotly* view can be displayed for zooming into data points. (iii) General plot setting tab. (iv) Export of a generated plot as a file in Portable Document Format (PDF).

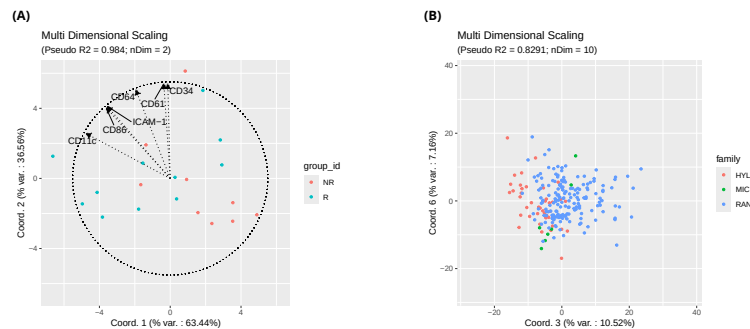


Figure 2. One customised plot of the *Krieg* dataset (A), and one of the *fsap* dataset (B) - exports from *MDSvis*.

Results

Cytometry use case

The main *MDSvis* use case is cytometry data analysis - notably QC and batch effect identification. In that case, the *CytoMDS* package is used to calculate sample EMD distances, based on cytometry marker expression matrices. An example of such a use case is described in the on-line package documentation (<https://uclouvain-cbio.github.io/MDSvis>). In the main vignette, the *Krieg_Anti_PD1* cytometry dataset [9] is used to demonstrate the different steps represented in Figure 1 (use case 1). The dataset consists of 20 baseline mass cytometry samples - prior to treatment - of peripheral blood from melanoma skin cancer patients subsequently treated with anti-PD-1 immunotherapy. The patient samples were split across 2 groups - non-responders and responders - and in 2 acquisition batches. The goal of the study was to identify biomarkers of responsiveness to immunotherapy at baseline. In Figure 1(B), a screenshot of the main *MDSvis* view panel (ii) shows one of several possible visualisations for this dataset - see also Figure 2(A) for a more visible version of this plot. Here, the user selected the first two axes for display and the dots, corresponding to each sample, are coloured according to the two patient groups. The user also chose to overlay a biplot, showing directions of marker medians correlated with the two axes, as well as a unit circle to facilitate quantitative visual assessment of correlations. Finally, a *plotly* interactive plot was activated - Figure 1(B) - for zooming in and displaying - by cursor - some additional sample specific information, like the sample full name, which would be too wide to display as labels on the main plot.

Extension to other MDS applications

The second use case consists in visualising a MDS projection, starting directly from any type of previously obtained sample distance matrix. It is illustrated in a second *MDSvis* package vignette, using the "frog skin active peptide" (*fsap*) dataset. This dataset consists of 226 samples, where each sample represents an amino acid sequence coding for a skin-secreted host defense peptide from a different species of amphibian (specifically, frogs). An ad-hoc distance matrix between these amino acid sequences was obtained by performing sequence alignment and computing an appropriate dissimilarity indicator (more detailed description is provided in the vignette). Based on this distance matrix, a MDS projection can be calculated by using *CytoMDS* and visualised by using *MDSvis*. One example of plot is shown in Figure 2(B). On that plot, axes 3 and 6 of the projection are represented, and dots are coloured by biological group (family). On that particular view, peptides belonging to the 3 different anuran families overlap while having different plot location distributions.

Discussion

Limitation and potential future extensions

The current main limitation of *MDSvis* is the need for the user to initiate the MDS projection calculation programmatically before importing the results into the Shiny application. A future extension could enable users to begin with a precomputed distance matrix and perform the MDS calculation directly within the application. This enhancement would also offer greater flexibility by allowing users to customise aspects of the MDS projection - for example, by selecting a subset of samples to be included in the projection, or, in the context of cytometry, specifying which markers should be used to compute sample distances. Another useful extension, for QC of cytometry samples, would allow the user to dig into a specific sample through dedicated sample views, for instance using *CytoPipelineGUI* [10].

Conclusion

In this work, we presented *MDSvis*, a Shiny application for interactive customised visualisation of MDS projections, aiming at facilitating data exploration. We described the analytic workflow, as well as the application features, which we illustrated by describing two use cases taken from the package vignettes. We believe that this software will facilitate access to visual exploratory data analysis by lowering the entry barrier, particularly for users with limited programming expertise.

Declarations

Funding

P.H. work is funded by GlaxoSmithKline Biologicals S.A., under a cooperative research and development agreement between GlaxoSmithKline Biologicals S.A. and de Duve Institute (UCLouvain).

Competing Interests

P.H. is a student at the de Duve Institute (UCLouvain) and participates in a post graduate studentship program at GSK. A.V. and L.G. report no competing interest.

Data Availability

The *Krieg_Anti_PD1* cytometry dataset is available from the *HDCytoData* Bioconductor R package. The *fsap* dataset is available from the *MDSvis* GitHub repository: github.com/UCLouvain-CBIO/MDSvis.

Software availability

The R package is available, under a GPL-3 license, from the *MDSvis* GitHub repository: github.com/UCLouvain-CBIO/MDSvis. The on-line documentation is available, under a CC-BY-4.0 license, from <https://uclouvain-cbio.github.io/MDSvis>.

Acknowledgments

The authors would like to thank Shabnam Zaman and Kim Roelants for providing the *fsap* dataset. This preprint was created using the LaPreprint template (<https://github.com/roaldarbol/lapreprint>) by Mikkel Roald-Arbøl .

References

- [1] Karl Pearson. "On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: [10.1080/14786440109462720](https://doi.org/10.1080/14786440109462720).
- [2] John A Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer New York, 2007. ISBN: 9780387393506. DOI: [10.1007/978-0-387-39351-3](https://doi.org/10.1007/978-0-387-39351-3).

- [3] Yvan Saeys, Sofie Van Gassen, and Bart N Lambrecht. "Computational flow cytometry: helping to make sense of high-dimensional immunology data". In: *Nat. Rev. Immunol.* 16.7 (July 2016), pp. 449–462. DOI: [10.1038/nri.2016.56](https://doi.org/10.1038/nri.2016.56).
- [4] Philippe Hauchamps et al. "Visual Quality Control With CytoMDS, a Bioconductor Package for Low Dimensional Representation of Cytometry Sample Distances". In: *Cytometry Part A* 107.3 (2025), pp. 177–186. DOI: [10.1002/cyto.a.24921](https://doi.org/10.1002/cyto.a.24921).
- [5] Darya Y Orlova et al. "Earth Mover's Distance (EMD): A True Metric for Comparing Biomarker Expression Levels in Cell Populations". In: *PLOS ONE* 11.3 (2016), e0151859. DOI: [10.1371/journal.pone.0151859](https://doi.org/10.1371/journal.pone.0151859).
- [6] Jan de Leeuw and Patrick Mair. "Multidimensional Scaling Using Majorization: SMACOF in R". In: *J. Stat. Softw.* 31 (Aug. 2009), pp. 1–30. DOI: [10.18637/jss.v031.i03](https://doi.org/10.18637/jss.v031.i03).
- [7] Winston Chang et al. *Shiny: Web Application Framework for R*. 2023. URL: <https://shiny.posit.co/>.
- [8] Carson Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN: 9781138331457. URL: <https://plotly-r.com>.
- [9] Carsten Krieg et al. "High-dimensional single-cell analysis predicts response to anti-PD-1 immunotherapy". In: *Nat. Med.* 24.2 (Feb. 2018), pp. 144–153. DOI: [10.1038/nm.4466](https://doi.org/10.1038/nm.4466).
- [10] Philippe Hauchamps et al. "CytoPipeline and CytoPipelineGUI: a Bioconductor R package suite for building and visualizing automated pre-processing pipelines for flow cytometry data". In: *BMC Bioinformatics* 25.1 (Feb. 2024), p. 80. DOI: [10.1186/s12859-024-05691-z](https://doi.org/10.1186/s12859-024-05691-z).

Chapter 10

CytoCNN: a CNN framework for predicting clinical outcomes using cytometry data

10.1 Introduction and context

This chapter presents the results of computational experiments done with deep [CNN](#) models (see Section [3.3](#)) to predict clinical outcomes of interest, using cytometry data as input. The motivation for this work package stemmed from the reading, in the early months of our PhD, of an article published in 2020 by Hu et al. [\[1\]](#). In this article, the authors propose and test a deep [CNN](#) architecture for classifying large sets of cytometry data samples, without preliminary identification of cell populations. By using convolutional layers in their model network architecture, they actually make an analogy between the single cell cytometry data matrices, and digital images, which is an innovative usage of [CNNs](#)^{[1](#)}.

More specifically, the authors took the sample files pertaining to nine large mass cytometry studies from the ImmPort database [\[3\]](#). The aim of these original studies was to assess the efficacy of different flu vaccines, and together they accounted for 472 healthy human [PBMC](#) samples. The authors trained

¹It should be noted that this article is not the earliest work to use [CNNs](#) for biomarker discovery from cytometry data. The authors themselves reference an earlier model, CellCNN [\[2\]](#).

their model on the [CyTOF](#) data to diagnose latent CytoMegaloVirus ([CMV](#)) in study participants, [CMV](#) status being one of the systematically collected patient related data in those studies. As one of the main article results, the evaluation metrics obtained for their model were impressive. For instance, the authors reported over 90% [AUC](#) in test (see [Section 2.6.1](#)), consistently across different choices of training, validation and test datasets [\[1\]](#).

Beyond those promising results, the methodology described in the article is also appealing due to a number of factors [\[1\]](#):

- Aside from the use of a [CNN](#) model - which can be considered relatively complex - the proposed approach itself is fairly simple, yet appears to be effective compared to other state-of-the-art methods. For example, it seems to outperform *FlowSOM* [\[4\]](#) (for cell population related feature extraction) combined with a random forest model (for classification), a methodology similar to the *FloReMi.1* model [\[5\]](#), which was the best method in the *FlowCapIV* challenge (see [Section 10.4](#)).
- The model input requires only minimal preprocessing of the data - compensation, scale transformation and ‘[QC](#) in time’ - and avoids a series of preprocessing steps like batch correction, and does not rely on population identification algorithms. Instead the model aims at identifying a direct association between raw cytometry data and the clinical outcome of interest, thanks to built-in feature extraction capabilities. This approach is illustrated in [Figure 10.1](#).
- As discussed in the article, the model seems to correctly filter out the study (batch) effects that were present in the [CyTOF](#) raw data. This suggests a model ability to combine heterogeneous datasets from several clinical studies, as soon as these datasets have a reasonable overlap of their cytometry marker sets.
- The model architecture provides some flexibility to include additional individual related covariates, e.g. age, gender, ethnicity, etc. - through additional nodes in the fully connected part of the [CNN](#) (see [Section 3.3.2](#)), to possibly improve the predictive capabilities of the model.
- A robustness analysis, consisting of a gradual decrease of the number of training samples, shows that about 200 training samples are needed to keep the high model predictive capabilities - which is reasonably low. Decreasing further the number of training samples leads to a gradual worsening of the [AUC](#) in test.

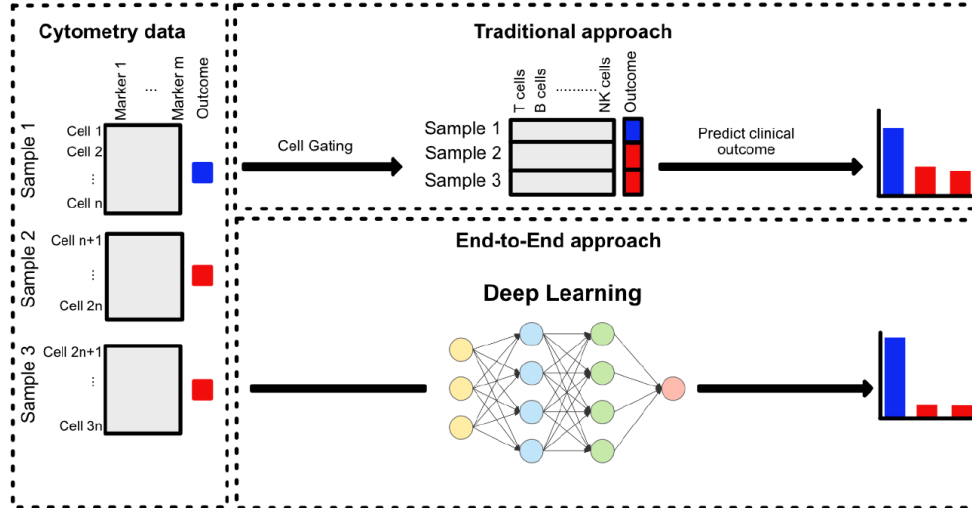


Figure 10.1: High level comparison between two main approaches of building cytometry data prediction models. Traditional approaches (top half) rely on gating methods (manual or semi-automated) to extract cell population information, and cell population statistics serve as features for downstream statistical models. In contrast, a deep learning approach based on [CNNs](#) can work with raw cytometry data with minimal preprocessing, thanks to the built-in feature extraction capabilities of the network. Figure reproduced from [\[1\]](#).

- Most importantly, as [ANNs](#) are intrinsically black box models, the authors attempted to build some model interpretation, by using eXplainable Artificial Intelligence ([XAI](#)) techniques [\[6\]](#). As a result, a non standard immune cell population was identified, which seems to be highly associated with the model [CMV](#) status prediction.

In view of these reported results, we decided to investigate the use of similar [CNN](#) models for other use cases and datasets. In what follows, we name *CytoCNN* our implementation of a Python framework, allowing to test various [CNN](#) architectural features and to predict outcomes of various types, with different input datasets.

The remainder of this chapter is structured as follows. Section [10.2](#) provides more details on the *CytoCNN* framework, its architecture and hyperparameters. In Section [10.3](#), a proof of concept and initial use case is described, where we try to reproduce some of the results obtained in [\[1\]](#). Then, Sections [10.4](#) and [10.5](#) are devoted to the two main computational experiments, i.e. predicting a survival outcome using the *FlowCapIV* dataset, and predicting

the age of human donors using the *GSK Immunosenescence* dataset. Finally, Section 10.6 sketches some conclusions of this work.

10.2 The *CytoCNN* model framework

In this section, we provide some details of the architecture and building blocks of *CytoCNN*, our own implementation of a deep CNN model as in [1]. Software pieces were programmed in Python, using *Keras* on top of *TensorFlow* for training and testing of the neural network. These Python modules have the advantage of providing built-in technical gateways to Graphical Processing Units (GPUs), enabling massively parallelised computation. Thanks to this technical infrastructure, we could run our experiments on a GPU-enabled High Performance Computing (HPC) platform, including a few Nvidia Tesla A100 GPU cards, which allowed us to perform substantially more computational runs than what would have been conceivable on a local computer.

Overall architecture

The high-level *CytoCNN* architecture is illustrated in Figure 10.2. Cytometry marker intensities are input to the convolutional layers, which constitute the feature extraction part of the network. The cell specific output activations of these layers are then summarised in pooling layers, and subsequently merged into a vector of sample features (blue part of the merge layer in Figure 10.2). Optionally, at this stage, additional sample characteristics (covariates) can be included as additional features, using additional nodes in this layer (in green colour on Figure 10.2). Then, the merged features serve as input to the dense, fully connected layers, which implement a non-linear prediction model. The activation of the output layer is the model prediction. Note that, depending on the use case, the *CytoCNN* framework is able to deal with classification, regression or survival problems (see further sections). The number of convolutional layers and their feature maps, as well as the number of dense layers and their neurons, can all be customised. Other hyperparameters are also configurable (see below for a detailed list).

Input cytometry data

As shown on Figure 10.2, input data needs to be presented to the network one sample at a time, in the form of a matrix of per cell marker intensities, with cells in rows, and markers in columns. To avoid providing too much noise to the network, these data matrices need to be minimally preprocessed, i.e. compensated, scale transformed and filtered for unstable signal parts ('QC in

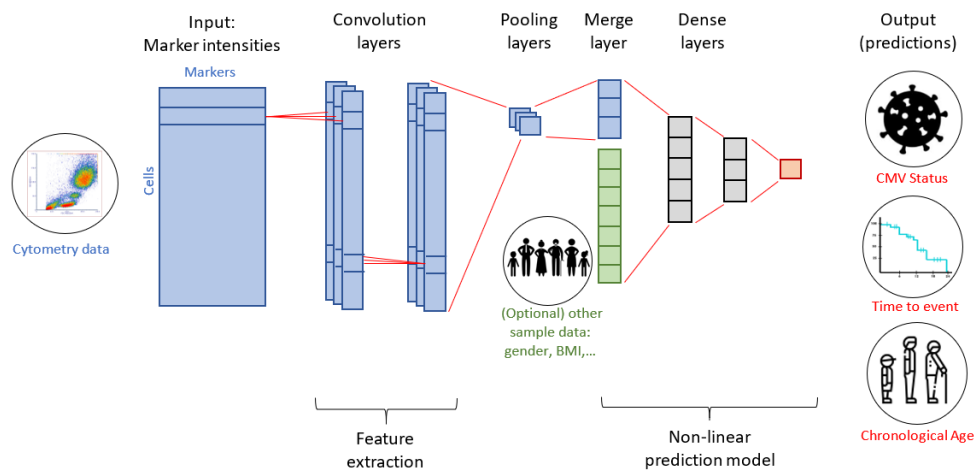


Figure 10.2: *CytoCNN* architecture.

time’, see Chapter 4).

An important consideration is input size, more specifically the number of rows, corresponding to cytometry events. First, the [CNN](#) needs a fixed size input layer, regardless of the particular cytometry sample presented as input to the network. Second, the forward and backward propagation processes require a single sample to completely reside in Random Access Memory ([RAM](#)). This effectively sets an upper bound on the number of rows the network is able to process, especially when using [GPUs](#). Unfortunately, marker intensities of millions of cells, as provided by modern flow cytometers, cannot be directly ingested when using the hardware infrastructure at our disposal, and therefore, a random subsampling of the cytometry data needs to be performed. In what follows, this subsampling was performed by randomly selecting entire rows from the cytometry data matrices, thus retaining the complete set of marker intensities for each selected cell and preserving the correlation between these intensities. In order to reach a trade-off between manageable memory size and keeping enough useful information to possibly identify relatively rare cell populations associated with biological signal, a number of cells in the range of 10,000 to 100,000 has to be used.

Convolutional layers

The convolutional layers in the *CytoCNN* architecture, inspired by [1], differ significantly from typical [CNN](#) layers used in computer vision (see Section

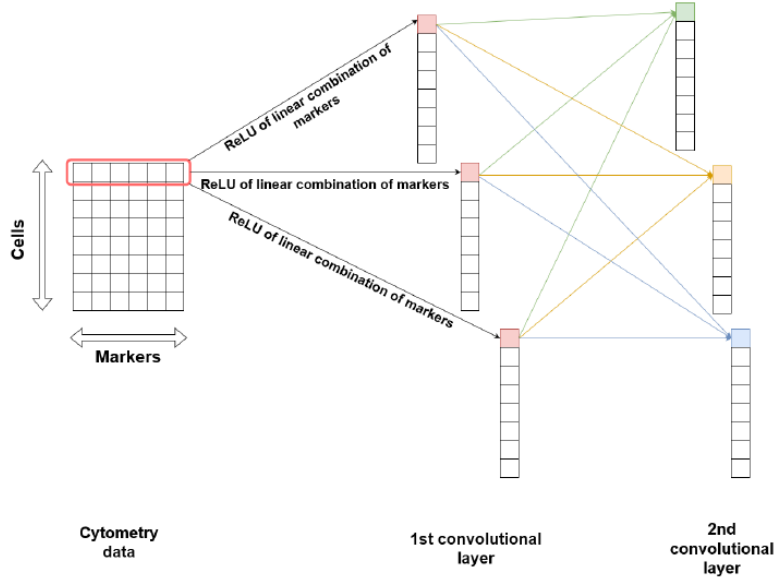


Figure 10.3: *CytoCNN* convolutional layers. Here, two convolutional layers, using three feature maps, are represented. Same colour arrows implies that the same network weights are applied by construction. Figure reproduced from [8].

3.3.2.1) in that they are 1D. Indeed, a key characteristic of cytometry data is that it represents an unordered collection of cells, i.e. there is no biological meaning attached to the fact that two rows are close to each other or not. Therefore, the neural network needs to be invariant to the permutation of rows in the data [7]. Moreover, there is also no meaning attached to the order of markers, hence the network should also be invariant to the permutation of columns. Together, these two constraints are satisfied by the choice of ‘single cell’ filters in the first convolution layer, which combine all marker intensities within the same row, but not across rows (Figure 10.3). As a consequence, the subsequent convolutional layers need to use a 1 by 1 filter. This means that each cell feature map value is a linear combination of the activation values of the previous layer, for that particular cell, and the same linear combination is applied for all cells (Figure 10.3). To introduce non linearity in the feature extraction, ReLU activation functions are used in each convolutional layer. Finally, to enhance network training stability and convergence, a batch normalisation layer (see Section 3.3.2.4) is added after each convolutional layer.

Pooling layer

Due to the particular structure of the convolutional layers, and also to preserve

the invariance to the permutation of rows - as discussed above - *CytoCNN* has only one pooling layer, summarising the convolutional activation value of all cells at once. As in [1], we prefer to use an average pooling layer, over a max pooling layer. Indeed, in the latter case, the output of the feature extraction part of the network would be effectively calculated based on the marker intensities of one single cell only, which would not make sense.

Dense layers

As shown in Figure 10.2, *CytoCNN* dense layers are standard fully connected layers. However, the activation function of the output layer needs to be customised, depending on the type of outcome (see Section 2.4). For standard regression and survival prediction, a linear activation function is applied. For binary classification, a sigmoid activation function is applied, forcing the prediction to lie between 0 and 1. In that latter case, the output value represents the predicted probability of positive outcome.

Optimisation algorithm

As mentioned before, training of the CNN is performed with the *keras* Python module, running on top of *TensorFlow*. As optimiser, we chose the Adam algorithm, of which the key tuning hyperparameter is the learning rate (see Section 3.2.3).

Comparison with Hu et al. [1] CNN model

The *CytoCNN* architecture is largely inspired by Hu et al. [1]. The most significant difference is its flexibility to predict different types of outcome (regression, classification or survival). Notably, handling survival outcomes with deep learning models has its own challenges and is still a research topic [9]. Specific *CytoCNN* implementation choices for survival outcomes will be discussed in Section 10.4.

On top of the ability to ‘plug’ different types of outcome, *CytoCNN* provides additional flexibility - in comparison with the model used in [1] - with the following functionalities:

- ability to perform validation using a fixed validation set, or using cross-validation
- ability to easily customise the network architecture (number of layers, number of feature maps, number of neurons per layer,...)
- ability to include regularisation, dropouts, or a combination of these

- ability to use some forms of context dependent data augmentation
- ability to read parts of the training set ‘on the fly’ in order to cope with possible [RAM](#) limitations

Hyperparameters and design decisions

Finally, here below are the design decisions and main hyperparameter choices to be made with *CytoCNN*:

- setting the input number of rows, i.e. the number of cells to randomly subsample in each cytometry sample - between 10,000 and 100,000, as mentioned before
- setting the number of convolutional layers, and their number of feature maps
- setting the number of dense layers, and their number of neurons
- using L1 or L2 regularisation, with respective choice of λ (see [Section 3.4.2](#)), or no regularisation at all
- using dropout layers, including a choice for the dropout rate (see [Section 3.4.3](#)), or not
- using data augmentation, with choice of specifically associated hyperparameters - depending on data augmentation technique, or not
- choosing the learning rate and the minibatch size (see [Section 3.2.3](#))

Depending on the context and problem at hand (see [Sections 10.3](#), [10.4](#) and [10.5](#)), these hyperparameters will be set, either by using grid search for optimising a validation metric (see [Section 2.5](#)), or simply by hand.

10.3 Proof of concept: predicting CMV status using mass cytometry data

As a first use case, we attempt to reproduce some of the results in [\[1\]](#). We therefore replicated, within the *CytoCNN* framework, the deep [CNN](#) architecture detailed in the article. Using the same input dataset, we then assessed whether we could obtain values of model evaluation metrics comparable to the article results. To facilitate this task, we could also take advantage of the

Python code provided by the authors themselves in a dedicated Github repository², which is in the form of a user tutorial. This allowed us to grasp some of the nitty-gritty implementation details, including low level methodological decisions.

Dataset and exploratory data analysis

We downloaded the data files provided by the authors in their Github repository, which contains data for 472 mass cytometry human PBMC samples from nine studies (see Section 10.1). The CyTOF data were already preprocessed, i.e. (1) scale transformed using arcsinh transformation ($y = \text{arcsinh}(x/5)$), (2) merged by keeping only the 27 markers that were present in all nine studies, and (3) downsampled to 10,000 cells from each sample [1]. We also collected the study identifier and the CMV status (positive or negative) for each sample. As in [1], we did not apply any batch correction to correct for heterogeneity between the 9 studies.

Some exploratory data analysis plots using *CytoMDS* [10] are shown on Figure 10.4. On the left plot, where data points are coloured according to CMV status, one is unable to discern any tendency of data point grouping by CMV status³. In contrast, the right plot, where the data points of the same MDS projection have now been coloured by study, clearly highlights study effects. This means that the biological signal associated with CMV status, if any, is probably small in comparison to the study associated sample heterogeneity. This was also mentioned in [1].

Results

Note that in what follows, and for the remainder of this chapter, a specific colour convention for result plots is used, i.e. training data results in red, validation data results in green and test data results in blue (whenever applicable).

We trained *CytoCNN* models, having the same architecture and using the same hyperparameters as in [1], i.e. learning rate set to 0.0001, and minibatch size set to 60 (see Section 3.2.3). We used the binary cross-entropy (Equation 3.5) as loss function, and allocated all data from one specific study to the fixed

²<https://github.com/hzc363/DeepLearningCyTOF>

³Other 2D axes combinations (not shown here) do not show more separation between the two groups.

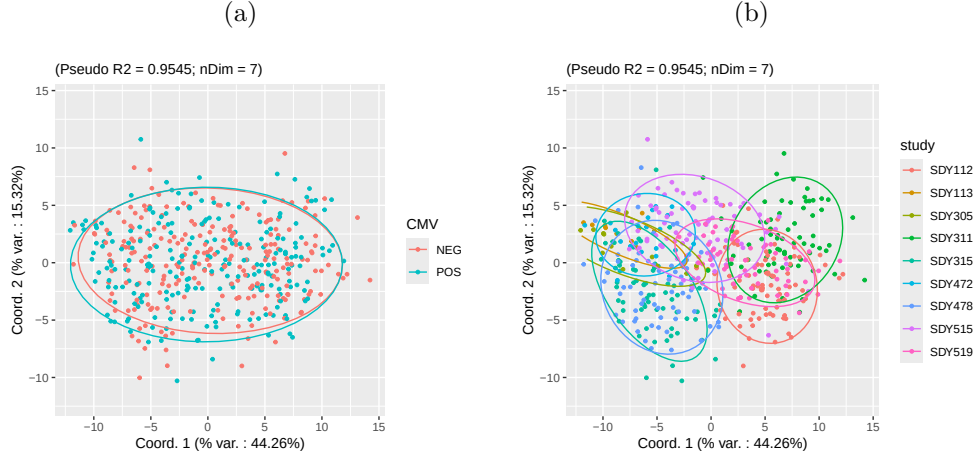


Figure 10.4: Exploratory data analysis using *CytoMDS* for the CMV dataset (axes 1 and 2 are displayed). (a) Data points are coloured according to **CMV** status: there is no visible grouping by **CMV** status. (b) Data points are coloured according to study: study effects are clearly visible.

validation set, and another study to the test set (again, same choices as in [1]).

However, we ran the training loop up to 5,000 epochs (instead of 2,000 epochs in [1]), and also repeated this model training process several times with different initial random seeds. Indeed, as the **CNN** weights are initialised randomly at the start of the training, different random seeds lead to different optimisation paths in the parameter space, hence to different models at the end of the training. It is therefore important to assess the variability encountered when performing the training loop with different initial random seeds, which gives an idea of the reproducibility of the process. Note that each training run, with 10,000 input cells and 5,000 epochs, took approximately 30 minutes on a A100 **GPU**.

Figure 10.5 shows, for a specific random seed - i.e. 42, the evolution of the training loss (a), the **AUC** in training (b), the validation loss (c), and the **AUC** in validation (d). The training loss decreases steadily as a function of the epoch and begins to converge when reaching 5,000 epochs, while the **AUC** metric increases and slowly tends to 1. In contrast, the validation loss starts by decreasing - after a very short initialisation phase - but reaches a minimum slightly before 2,000 epochs, and then increases sharply while showing large

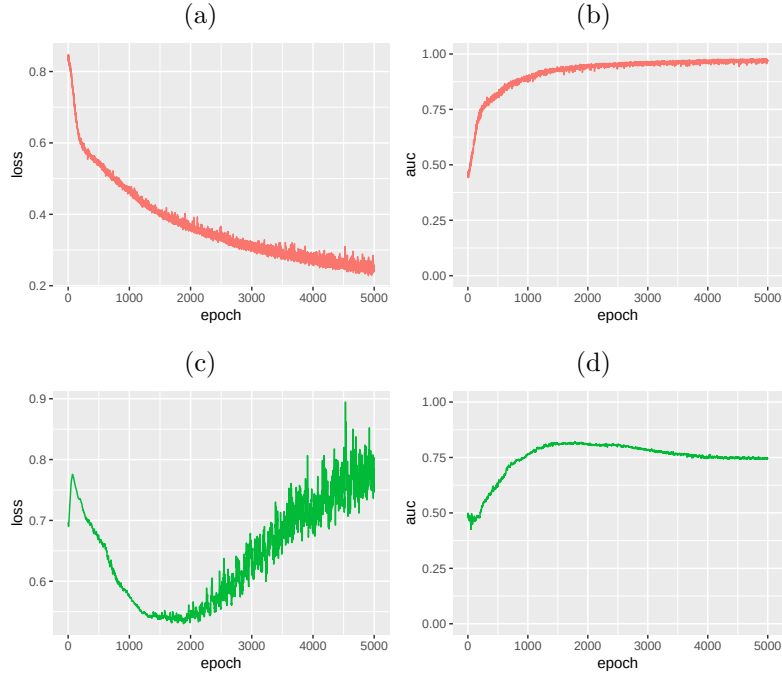


Figure 10.5: Learning curves obtained with a specific model weight initialisation, i.e. random seed = 42: the training loss (a), the [AUC](#) in training (b), the validation loss (c), and the [AUC](#) in validation (d).

oscillations. In parallel, the [AUC](#) in validation shows a symmetrical behaviour to the one of the validation loss, starting with a sharp increase, then reaching a plateau and slowly decreasing. This is a typical model overfitting phenomenon, as discussed in Section 3.4.1. Figures D.1 to D.4, displaying these 4 curves for 10 different random seeds, show that similar overfitting is observed for each training run, although the exact shape of the validation curves can be somewhat different, and the maximum [AUC](#) reached in validation differs from run to run.

To cope with overfitting, the authors of [1] applied a variant of early stopping consisting in storing the best model, i.e. having the lowest loss in validation, obtained during the training phase - see Section 3.4.1. We implemented the same procedure, obtained the best model predictions on the test set, and calculated the corresponding [ROC](#) curves (see Section 2.6.1), both in validation and test.

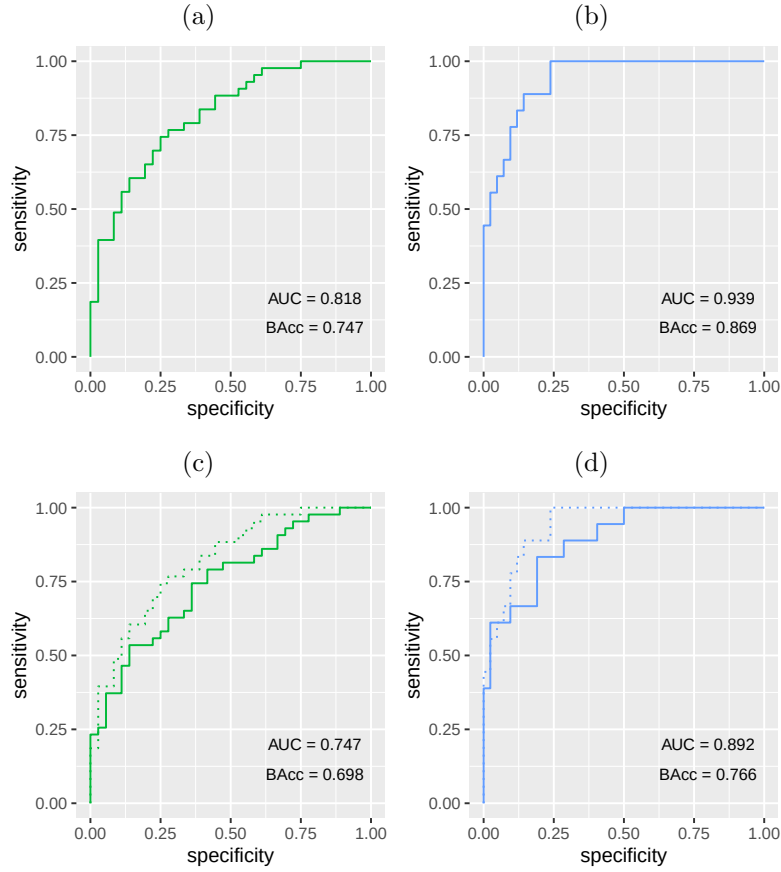


Figure 10.6: ROC curves obtained in validation and test for random seed 42. On the top row, the validation (a) and test (b) ROC curves are shown for the best model in validation obtained during the training phase. On the bottom row, the validation (c) and test (d) ROC curves are shown for the final model obtained at the end of the training, after 5,000 epochs, showing deteriorated evaluation metrics as compared to the best model in validation of the top row (corresponding dashed ROC curves are overlaid on the plots).

The top row of Figure 10.6 shows the ROC curves obtained for random seed 42 and applying the procedure described above, both for the validation (a) and test (b) sets. These curves have been annotated with the corresponding AUC, as well as the BAcc value obtained, when using the best possible threshold in validation. Figures D.5 and D.6 provide the same two curves for 10 different random seeds. The following conclusions can be drawn from the inspection of these curves:

- For random seed 42, the obtained test evaluation metric is comparable to what is reported in [1], i.e. a AUC of about 0.94 (Figure 10.6).
- Not all different random seeds lead to such good results in prediction (see Figure D.6). For instance, seeds 45 and 47 lead to poor models with an AUC being only slightly better than an average random prediction.
- In most run instances, the test metric is notably higher than the validation metric. We assume that this is probably due to the characteristics of the samples chosen for the fixed validation and test sets, i.e. the test outcomes being easier to predict than the validation ones. This was verified by swapping the studies used for validation and test, respectively. In that latter case, the validation metric becomes higher than the test metric (results not shown here).
- The model instance having the smallest ever validation loss - among all models - is for random seed 43 (see Figure D.3), which leads to a test AUC of around 0.9 - not the best AUC in test thus.

The bottom row of Figure 10.6 shows the validation (c) and test (d) ROC curves obtained for random seed 42, but this time using the model as obtained at the end of the training - after 5,000 epochs. Here, the evaluation metrics, both in validation and test, are clearly deteriorated compared to the best model in validation of the top row - with the corresponding dashed ROC curves overlaid on the plots. This highlights how important it is to control for model overfitting.

Conclusion

As a conclusion to this proof of concept section, we demonstrated the usability of our *CytoCNN* framework, and succeeded in obtaining similar classification metrics as in [1], on the same dataset and with the same CNN architecture. Besides, by examining multiple models obtained with different training runs, we could draw the following additional conclusions:

- Different random seeds lead to different model instances, which can lead to sometimes very different predictive capabilities.
- It is paramount to control for model overfitting, as here, keeping the best model encountered during training - in terms of validation loss - was key to reach better predictive results.

10.4 Predicting time to progression to AIDS for HIV+ patients

In this section, we leverage the *CytoCNN* framework to build a predictive model for a survival outcome. In particular, we use the *FlowCapIV* dataset to predict time of progression to Acquired ImmunoDeficiency Syndrome (AIDS) for Human Immunodeficiency Virus (HIV) seropositive patients.

10.4.1 *FlowCapIV* challenge and dataset

The *FlowCapIV* dataset comes from a challenge that was organised by the Flow Cytometry: Critical Assessment of Population identification methods (FlowCAP) consortium. Set up in the first half of the 2010's, FlowCAP organised competitions for researchers and bioinformaticians, with as goals “to advance the development of computational methods for flow cytometry data analysis by providing the means to objectively test and compare these methods, and to provide guidance to the end user about how best to use these algorithms.” [11]

With the *FlowCapIV* challenge, organised in 2015, participants were asked to analyse flow cytometry data from a HIV clinical study [12]. The goal was to identify immune correlates associated with a specific clinical outcome: the time to progression to AIDS in HIV-positive individuals. Indeed, infection with HIV typically unfolds in two major stages. The first stage, known as seroconversion, corresponds to the point at which the virus becomes detectable in the bloodstream and the immune system starts producing antibodies against it. In this point, the patient is declared HIV-positive. The second stage, the onset of AIDS, may occur months or even years later and is characterised by a profound weakening of the immune system, which leaves the individual vulnerable to opportunistic infections and certain cancers. In the study, a cohort of 383 individuals with well-documented HIV seroconversion dates and long-term clinical follow-up was analysed. For each participant, the time to AIDS diagnosis, i.e. disease progression - referred to as the survival time - was either

observed directly or treated as a censored event, for instance when antiretroviral therapy was initiated or when the individual was lost to follow-up. This design corresponds to a typical time-to-event or survival analysis framework, in which both observed and censored outcomes are jointly modeled to identify biological markers associated with disease progression.

Then, for each individual, two samples were provided: (1) a **PBMC** sample stimulated with HIV-Gag peptides - herebelow referred to as the ‘stimulated’ sample - measuring a possible immune response to one of the **HIV** antigen, and (2) an unstimulated **PBMC** control, giving indication of the patient baseline immune state - herebelow referred to as the ‘unstimulated’ sample. For each sample, FSC-A, FSC-H, SSC-A and 13 fluorescent channel intensities were measured (corresponding to CD3, CD4, CD8, V-Amine/CD14, CD27, CD107-A, CD45RO, CCR7, CD57, CD154 markers and IFN γ , TNF α , IL2 cytokines).

The *FlowCapIV* participants were provided with the flow cytometry data for all patients, and the time to progression to **AIDS** - or censoring time - of a subset of 191 patients - the training set - and had to predict the outcome for the remaining 192 patients - the test set. Among the nine participating models, only two had model predictions significantly correlated with the true outcomes [12]. The best model was *FloReMi.1* [5], which used automatic cell population selection and creation of automatic cell population features, followed by selection of features with minimal redundancy, and a random survival forest model for survival prediction.

We chose the *FlowCapIV* dataset for experimenting *CytoCNN* for a number of reasons. Besides being a publicly available dataset, it also contains several hundreds of subjects, which is in line with the requirement to have more than 200 samples in the training set, as highlighted by the robustness study in [1] (see Section 10.1). Moreover, predicting a survival type of outcome using a **CNN** also leads to interesting methodological challenges - among these, selecting adequate loss function and evaluation metrics. Finally, the *FlowCapIV* challenge results showed that it was possible to find cellular correlates of progression to **AIDS** in the dataset, but the task proved to be challenging. Indeed, although the best model, *FloReMi.1*, obtained a 89% **C-index** in cross-validation - using the provided training set - it only got 67% **C-index** on the test set [5].

The challenging nature of the prediction task can be confirmed by ex-

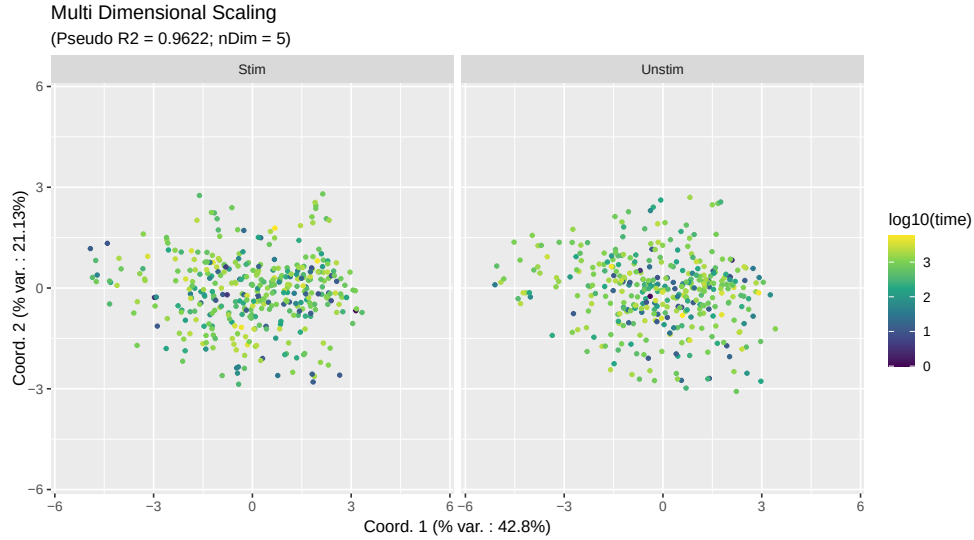


Figure 10.7: [MDS](#) projection of the *FlowCapIV* samples, using the first two axes - , on the left, the stimulated samples, and on the right, the control unstimulated samples. Sample dots are coloured according to time to event (expressed in $\log_{10}$). No clear pattern can be identified that would associate the sample positions on the plot with the time to event. Also, the distributions of stimulated and unstimulated samples are similar.

ploratory data analysis, as shown in Figure 10.7. On these plots, preprocessed (see Methods) cytometry data samples are represented in a low dimensional projection using *CytoMDS* [10]. This plot does not reveal clear patterns associated with higher or lower time to progression to [AIDS](#).

10.4.2 Methods

We downloaded the *FlowCapIV* dataset, i.e. 766 [FCS](#) files and sample meta-data, from Flow Repository [13] (repository ID: FR-FCM-ZZ99) and preprocessed the flow cytometry data using *CytoPipeline* [14]. The preprocessing steps applied were the following:

- margin removal using *PeacoQC* [15];
- compensation using the compensation matrix provided within the [FCS](#) files;
- ‘[QC](#) in time’, i.e. eliminating events belonging to parts of unstable signal, using again *PeacoQC* [15];

- doublets removal using *flowStats* [16];
- scale transformation using logicle transformation [17] for fluorescent channels - with automated calibration using *flowCore* [18], and linear transformation for scattered light channels - aligning 5 and 95 quantiles of *FSC/SSC* channels to those of a reference fluorescent channel.

Then, we implemented *CytoCNN* models to predict progression to AIDS of HIV+ patients. As the outcome is a survival time, we applied a linear activation in the output layer, and used the negative partial log-likelihood as a loss function (Equation 3.6). For the evaluation metric, we chose the unbiased version of the C-index (see Section 2.6.3.3).

In order to have enough training data, we did not put ourselves fully in the conditions of the *FlowCapIV* challenge participants - who had only access to the outcome for half of the patients - but instead allocated 300 patient samples for training. The remainder - samples belonging to 83 patients, hence 21.7% of the samples - were allocated to a fixed test set. To get a good grasp of the generalisation capabilities of the models, and given the limited size of the training set (when taking censorship into account - see sample size discussion in Section 10.4.4), we implemented a 4-fold cross-validation loop (see Section 2.5). In line with the conclusions of our initial proof of concept, we systematically kept the best model seen during training, from the validation loss viewpoint, and took that particular model for our performance measurements in validation and test.

Hyperparameter values were set according to the following choices:

- the input size was set to 10,000 randomly sub-sampled cells. A limited test with 50,000 cells did not show noticeable result changes, and needed much more RAM, limiting the number of possible runs in parallel.
- the number of convolutional layers was set to 2, with 3 feature maps. The fully connected part of the network consisted of the merge layer with 6 nodes (see below), one single intermediary dense layer with 3 neurons, and the output layer with 1 neuron. Note that the size of this network architecture can be considered as minimal, given the chosen CNN model framework (Figure 10.2). In fact, adding more layers and/or neurons in the architecture only tended to worsen the overfitting problems encountered during model validation (see Results section). Also, we did not add any additional patient specific covariate in the merge layer.

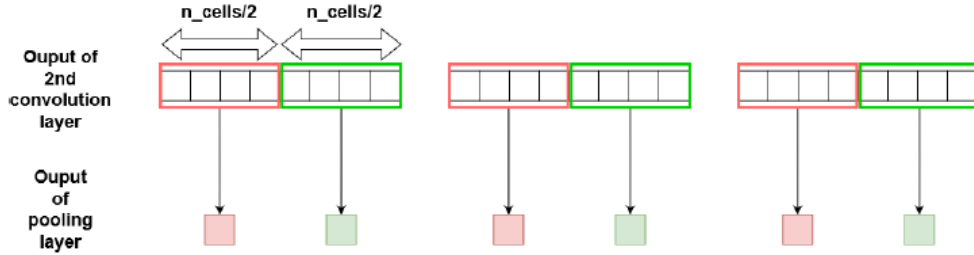


Figure 10.8: Schematic representation of the *CytoCNN* pooling layer, for the *Flow-CapIV* dataset. Red parts represent feature values related to unstimulated cells, green parts are values related to stimulated cells. Figure adapted from [8].

- the learning rate was tuned using grid search (see Section 2.5), and finally set to 0.001.
- since the negative partial log-likelihood loss can only be calculated using the data of the whole training set at once (Equation 3.6), each epoch has only one minibatch, or equivalently, the minibatch size needs to be set to the size of the whole training set.

Importantly, when designing the network architecture for this particular *CytoCNN* use case, we had to cope for a specific data characteristic, i.e. the fact that there were two different cytometry samples per patient, i.e. a *stimulated* one, and an *unstimulated* one. Therefore, we concatenated the 10,000 cell rows of both samples of each patient, making a 20,000 row input matrix per training data point. By doing this, we made sure, thanks to the convolutional layers, that the same features were extracted for cells belonging to both types of samples. This setting also had an impact on the design of the pooling layer. As depicted in Figure 10.8, given that the stimulated and unstimulated cells are concatenated, the average pooling needs to be divided into two parts: pooling for the stimulated cells and pooling for the unstimulated cells. The 3 feature maps, calculated for the two types of cells, are then gathered into a 6 node merging layer, which in turn serves as input to the fully connected part of the network.

Using these hyperparameter choices, we defined a ‘base’ model for which we did not use any regularisation, dropout layers or data augmentation. However, as we had to cope with massive overfitting (see Results), we had to build refined versions of the base model, resorting to varying techniques:

- for regularisation, we implemented the L2 version, tuned the related λ

hyperparameter value (Equation 3.13) by grid search (see Section 2.5), and finally set it to 1.0.

- for dropouts, we implemented the dropout layers for convolutional and dense layers, and set the dropout rate value to 0.4, which is a standard value according to the literature [19].
- for data augmentation, we designed an ad hoc procedure, which we describe below.

Ad hoc data augmentation procedure

Data augmentation, as one possible way of dealing with overfitting, was mentioned in Section 3.4.4. It consists in creating additional training data points, by duplicating existing training samples while introducing small perturbations on inputs, target values, or both.

To apply data augmentation in our specific context, we first define as hyperparameter, a data augmentation factor, i.e. an integer $n_{DA} > 1$, which is the number of times each training data point is to be duplicated. Second, to generate a new duplicate, we perform a new subsampling of 10,000 cells, drawn from the corresponding original sample. Third, while we keep the observed/censored status of each data point, we generate perturbations for the target values (observed survival time or censoring time). For that purpose, we noticed that the survival loss function is only sensitive to the order of times of the different training samples, and not to the time values themselves (see Equation 3.6). Therefore, we implemented a perturbation process that induces a slight modification of the order of times to event (or times to censorship). In other words, we assign to a duplicated training data point, not its original time, but another time, randomly selected among its close neighbours.

Formally, if we consider one duplicated training data point i , and designate by $k(i)$ its index in the vector of increasing ordered times for the whole training set, we allocate to this duplicated data point a new time to event by randomly selecting a new index $k^*(i)$ such that:

$$k^*(i) = k(i) + (-1 + 2Y)(1 + X) \quad (10.1)$$

where $X \sim \text{Poisson}(\mu)$ with $\mu = 0.5$

and $Y \sim \text{Bernoulli}(p)$ with $p = 0.5$

and $X \perp\!\!\!\perp Y$

In Equation 10.1, the second term of the RHS represents the (signed) index shift. The $(1 + X)$ factor sets an index shift absolute value of 1, 2, 3, $\dots$, while

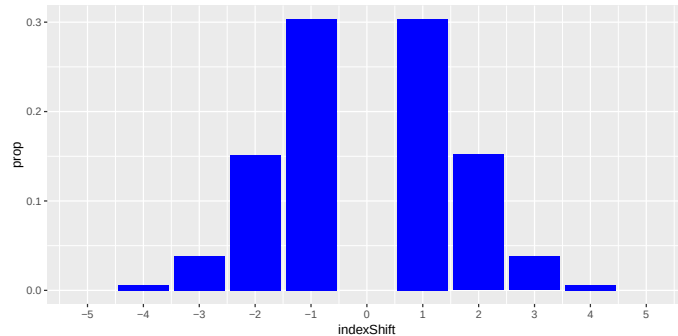


Figure 10.9: Histogram of the index shift obtained when applying the data augmentation procedure.

the $(-1 + 2Y)$ factor sets the index shift sign (+ or -). Applying this data generation process leads to a distribution of time index shifts as depicted in Figure 10.9, i.e. centered and symmetric around 0.

10.4.3 Results

10.4.3.1 Base model

Using the base model configuration described above, we trained several instances of *CytoCNN* model. We ran the training process for 2,000 epochs, with 4 different random seeds, and including 4-fold cross validation, we obtained in total 16 different model instances, which we could evaluate on the corresponding fold-specific validation set. With 20,000 input rows and 2,000 epochs, each 4-fold cross-validation loop took approximately 50 minutes on a A100 GPU. Learning curves for a specific seed and fold combination are shown in Figure 10.10. The (a) subplot shows the evaluation of the training loss, which decreases steadily with the epoch number. In the (b) subplot, the C-index in training increases gently and converges to a high value, near 1. However, the (c) subplot shows that the validation loss is completely erratic, does not even decrease in the beginning and instead, reaches huge values. The C-index in validation, on subplot (d), shows a corresponding erratic behaviour, and stays at a low value. Globally, this behaviour probably corresponds to a massive overfitting phenomenon, where the model seems to learn the training data well, but is completely unable to generalise on the validation set, to the point that we can say that the model is *fitting noise*, i.e. fitting details of the training set that are not representative of any biological signal of interest. Figures D.7 to D.10 reveal that the same erratic behaviour is observed for all

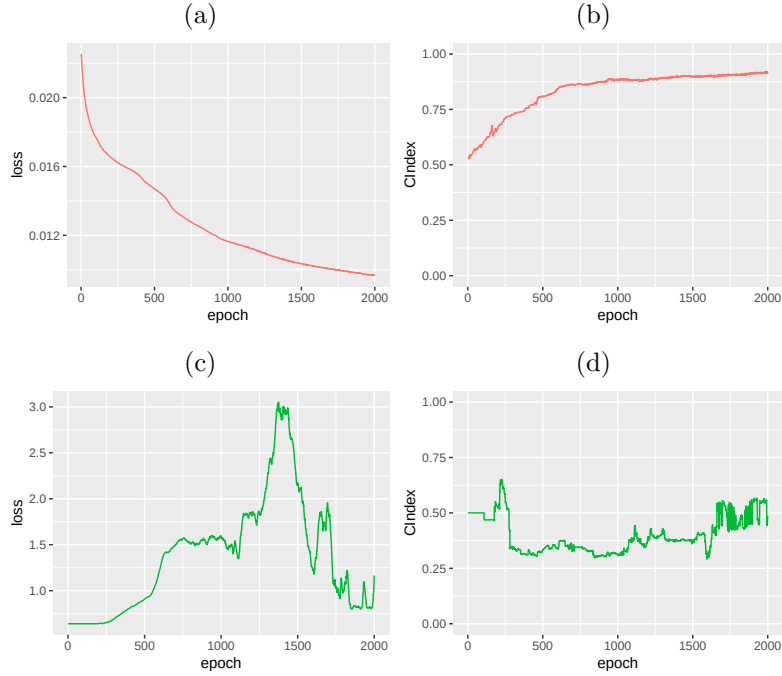


Figure 10.10: Learning curves obtained with the base model. Here curves for random seed = 42, and fold = 0 are shown: the training loss (a), the training C-index (b), the validation loss (c), and the validation C-index (d).

model instances (seed/fold combinations).

Diagnostic plots

To explain the previous plots, and visualise individual data point predictions, we designed a diagnostic plot, showing the evolution of the model predictions as the training process progresses. Figures 10.11 and 10.12 show examples of such plots, for one random seed and all 4 folds, for training and validation, respectively. Since we are dealing with survival data, model predictions take the form of risk scores (see Section 3.2.1.3). In each subplot, these risk scores are plotted against the true data point times (event observation or censoring time). In such plots, an ideal model prediction - corresponding to a C-index equal to 1 - is a situation where (1) risk scores attached to observed events (purple dots) are monotonically decreasing as a function of true observed time, and (2) risk scores attached to censored data (green dots) lay

inside the envelope created by the decreasing purple dots. On these figures, each row of subplots corresponds to the evolution of predictions for a specific fold, and each column corresponds to a specific epoch during the learning process.

Looking at the training evolution (Figure 10.11), one can clearly see that, as the number of epochs increases, the model predictions clearly evolve toward the ideal model prediction. However, digging onto the same plots in validation (Figure 10.12), one has to realise that, although the risk scores move quickly away from their small initialisation values, there is no convergence of patterns observed in the prediction when the number of epochs increases. To the contrary, the model validation predictions jump from one pattern to another, showing the incapacity of the model to identify a true biological signal.



Figure 10.11: Model predicted risk scores for the training set data points (observed event points in purple, censored points in green). Each row corresponds to one fold, random seed = 42, increasing epoch number from left to right. Corresponding C-index are overlaid on each subplot.



Figure 10.12: Model predicted risk scores for the validation set data points (observed event points in purple, censored points in green). Each row corresponds to one fold, random seed = 42, increasing epoch number from left to right. Corresponding C-index are overlaid on each subplot.

10.4.3.2 Attempts to deal with overfitting

In order to cope with the model behaviour described above, we implemented various techniques for dealing with overfitting, i.e. regularisation, dropout layers, and data augmentation.

Applying regularisation

Learning curves for a specific seed and fold combination, with L2 regularisation and $\lambda = 1.0$ (see Section 10.4.2) are shown in Figure 10.13 - subplots (a) to (d), and for all run seeds and folds in Figures D.11 to D.14. These plots are qualitatively very similar to those obtained in the previous section - for the base model - meaning that unfortunately, the overfitting is still massively present.

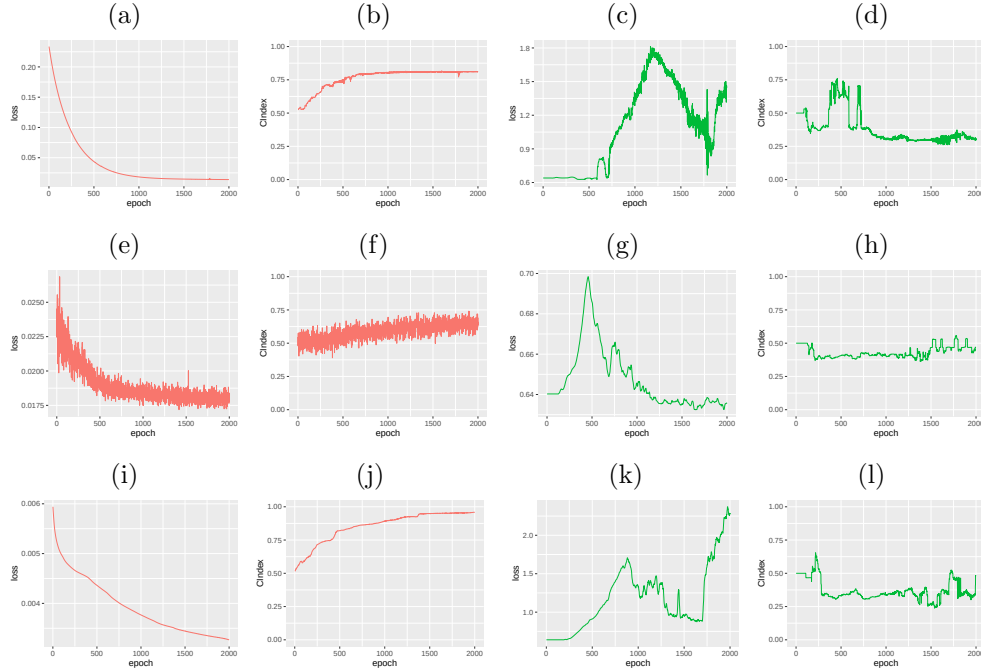


Figure 10.13: Learning curves obtained when applying different techniques to deal with overfitting. Here curves for random seed = 42, and fold = 0 are shown. In each row, we have, from left to right: the training loss, the training **C-index**, the validation loss, and the validation **C-index**. The first row - subplots (a) to (d) - corresponds to regularization; the second row - subplots (e) to (h) - corresponds to dropout layers; the third row - subplots (i) to (l) - corresponds to data augmentation.

Applying dropouts

We also tried to add dropout layers to the base model (see Section 10.4.2). Corresponding learning curves are shown in Figure 10.13 - subplots (e) to (h), and for all run seeds and folds in Figures D.15 to D.18. Here, the training loss curve is highly oscillatory and globally decreasing but slowly (e). This is combined with a near 0.50 **C-index** oscillating curve (f). This shows that the model has difficulty learning the characteristics of the training set. Note that when increasing the number of epochs to 5,000 (results not shown here), the training loss did not continue to decrease, leading to poor training results. Without surprise, this also leads to poor results in validation - subplots (g) and (h). As a conclusion, it seems dropout layers are not of any help in our case.

Applying data augmentation

Finally, we also applied data augmentation, as described in Section 10.4.2, and with a data augmentation factor n_{DA} set to 5. Unfortunately, this did not improve the overfitting problem, as the learning curves displayed in Figure 10.13 - subplots (i) to (l), and Figures D.19 to D.22 are very similar to the ones observed for the base model.

Combinations of techniques

We also tried to combine two or more of the techniques described above, but with no more success (results not shown here).

Notably, when initially using a combination of dropout layers with data augmentation, we suddenly got much better metrics in validation [8], for instance a C-index in validation of 0.75 or even more (not shown here). However, evaluating the obtained models using the independent test set could not confirm these good results in validation. After investigation, it turned out that this was due to a wrong implementation of the data augmentation procedure described in Section 10.4.2. Indeed, training set duplicates were generated beforehand, and the obtained extended training set was used as input to the 4-fold cross validation procedure, as if these data points were independent. By doing this, we were in fact creating a breach whereby some validation data information was used to generate synthetic data points, which in turn were incorporated in the training set, leading to the famous data leakage problem, as mentioned in Section 2.5. Therefore, we reviewed our implementation of data augmentation as to include it in the cross-validation loop. But then, this revised implementation again lead to poor generalisation, in line with Figure 10.13.

10.4.4 Discussion

The various plots shown in the previous section all show the same massive overfitting phenomenon. Why is that? In fact, the problem most probably lies in the limited amount of training data provided to the network. This is in apparent contradiction with the conclusions of the robustness study performed in [1] where it was found that 200 data samples were enough to obtain very good evaluation metrics in test, and the fact that we have here about 225 data in the training set of each model. However, in a survival problem, not all data point convey the same amount of information. Censored data only provide a lower bound of the true time to event, and therefore, the lower the censoring

time, the lower the amount of information contained in the data point. It turns out that, in the *FlowCapIV* dataset, a very high proportion of data is censored (about 80% of the patients). Moreover, most censoring times are low compared to the distribution of observed event times. This can be clearly seen in Figure 10.14, either on the Kaplan-Meier survival curves (a), or on the distribution of times per event type (b). This means that, translated into an ‘equivalent number of observed events’, each model could have as few as about 50 to 70 training samples as input, which is probably largely insufficient to fit a deep CNN model.

To wrap up this *FlowCapIV* experiment section, we can conclude that we technically managed to apply the *CytoCNN* architecture to survival data, by implementing risk factor predictions, negative partial log-likelihood as loss function, complemented with C-index evaluation metric, facilitating interpretation of model evaluation. However, with the *FlowCapIV* dataset, we came across a massive overfitting issue, which we could not improve by any of the various tested techniques. After investigation, we could reasonably attribute this unsuccessful result to the relatively small amount of information contained in the dataset, given the high rate of early censorship observed in the data.

Throughout this process, we learned two important lessons:

- exploratory data analysis and diagnostic plots are essential tools for interpreting model behaviour.
- data leakage can be introduced surprisingly easily when model training is coupled with preprocessing tasks like data augmentation.

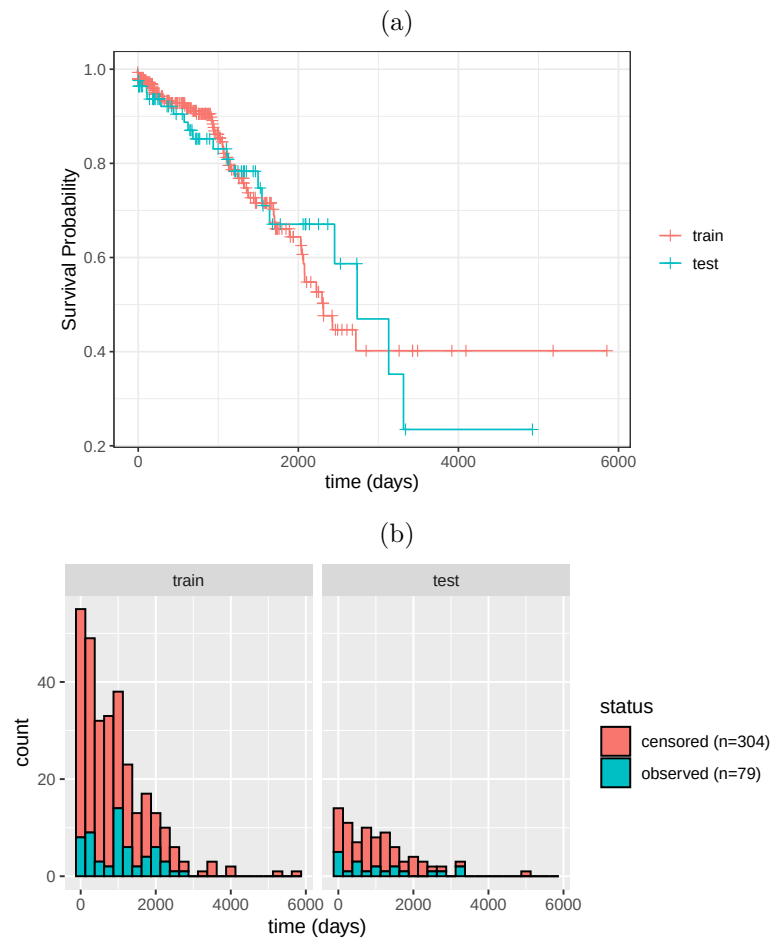


Figure 10.14: (a) *FlowCapIV* Kaplan-Meier survival curves, split between training and test sets. (b) Overlaid distributions of censored and observed times, split between training and test sets.

10.5 Predicting the age of individuals using flow cytometry data

10.5.1 Introduction

This section is devoted to our last use case of the *CytoCNN* framework, i.e. a regression problem where the goal is to predict the age of human individuals, based on flow cytometry data.

The motivation behind this exercise originated from a GSK study aimed at characterising *immunosenescence*, i.e. the immune system dysfunction or abnormal activation, predisposing individuals to age-related diseases [20]. Immunosenescence also translates into altered vaccine response in elderly [21], an effect that can sometimes be overcome by using potent adjuvants in vaccine formulations [22]. Therefore, for vaccine research and development, identifying potential biomarkers of immunosenescence is very interesting, as these could be used for patients' stratification during clinical trials as well as for adjuvant selection. However, one issue with immune age is that it is a concept that is not easily measured. Indeed, it is influenced by specific lifetime events, i.e. environmental conditions, lifestyle and past infections experienced by an individual during their lifespan [23]. In other words, immune age of an individual is only partially correlated with their chronological age.

Nevertheless, in the *CytoCNN* computational experiment described here, we used chronological age - an easily measured target variable - as a proxy for immune age. We reasoned that if our model would be capable of identifying general trends present in cytometry data of large cohorts of elderly - differentiating their immune system from younger ones - it could turn out that these cytometry data characteristics could also be valid biomarkers of immune age - although arguably, this would not be guaranteed.

10.5.2 Dataset

The dataset under study, referred to in what follows as the *GSK Immunosenescence* dataset, is based on the re-use of existing samples and data from four different completed past vaccine clinical studies, i.e. two zoster and two influenza vaccine studies. In each study, **PBMC** samples of participants had been collected prior to vaccination, for characterisation of the immune system at baseline, and part of these samples had been frozen for potential future re-use. Then, in the GSK Immunosenescence study, these samples were thawed and, after thorough sample **QC**, 485 of these could be acquired with a general

immunophenotyping flow cytometry panel comprising 26 different fluorescent channels (corresponding markers not detailed here). Data acquisition had to be split into 16 flow cytometer runs (batches). For technical reasons not detailed here, the sample of the same studies had to be acquired subsequently in the same batches, i.e. studies and runs could not be crossed - see also Figure 10.15, panel (e). To control for potential batch effects, a unique PBMC control sample, originating from a single healthy donor, was split in 16 sub-samples, and those 16 *bridge* samples were acquired in their corresponding batch, with the aim to highlight technical run-to-run variability.

On top of the cytometry data, some individual characteristics of the study participants were also collected: age, sex, ethnical group, CMV status (positive or negative), and vaccine response - derived from antigen specific antibody titers in blood serum, after comparison with corresponding antibody titers at baseline.

In Figure 10.15, subplots (a) and (b), the MDS projection of the flow cytometry data, performed using *CytoMDS* [10], highlights a tendency of younger individual samples to be located on the left side of the first axis. This seems to indicate that the flow cytometry data indeed contain some age related patterns. However, some additional exploratory data analysis plots - subplots (c) to (g), reveal two potential issues in the dataset. First, the distribution of individual ages is bi-modal, with a strong imbalance towards the older individuals. Second, 3 variables, i.e. age, original study and data acquisition runs, are confounded to various extents.

10.5.3 Methods

We preprocessed the FCS sample files of the 4 different studies, together with the *bridge* samples - 501 FCS files in total - using *CytoPipeline* [14]. The steps applied in the preprocessing pipeline were the following:

- margin removal using *PeacoQC* [15];
- compensation using run specific compensation matrices, manually adjusted by an expert scientist;
- ‘QC in time’, using again *PeacoQC* [15];
- doublets removal using *flowStats* [16];
- dead cells and debris removal, by using *FlowWorkspace* [24] to automatically apply manual gates defined in *FlowJo* by the expert scientist;

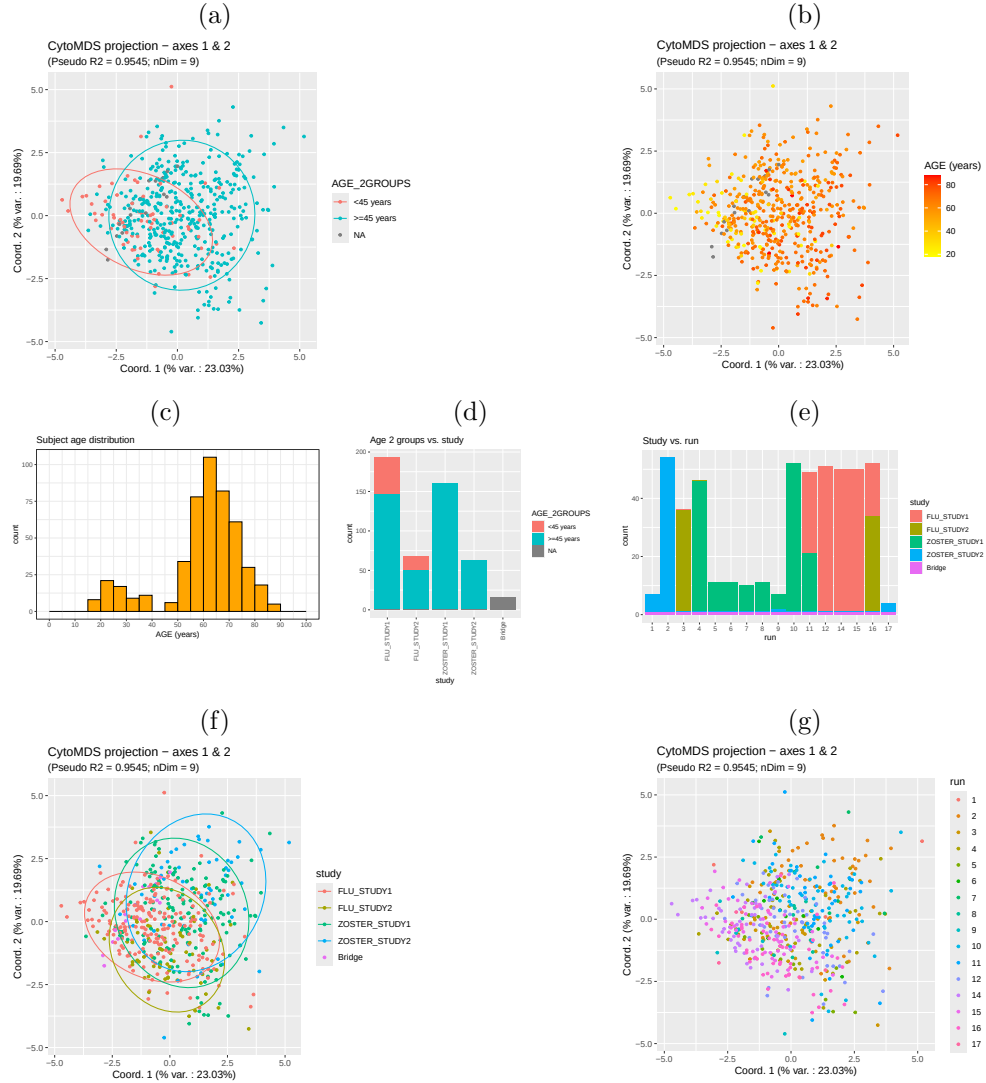


Figure 10.15: (a) *CytoMDS* plot of the *GSK Immunosenescence* dataset, with data points coloured according to the 2 age groups (below and above 45 years old). ‘NA’ grey dots correspond to the 16 *bridge* samples, i.e. batch control samples for each data acquisition run, for which the donor age is unknown. (b) Same plot, with data points coloured according to age, as a continuous variable. (c) Distribution of individual ages. The distribution is bimodal, with a strong imbalance towards older individuals. (d) Stacked barplots of age groups, split by study. Younger individuals are concentrated in two studies only. (e) Stacked barplots of study, split by data acquisition run. (f) *CytoMDS* plot, coloured by study, highlighting some heterogeneity between studies. (g) *CytoMDS* plot, coloured by run, highlighting a clear run effect. It is confounded with the study and, to a certain extent, the age group (a).

- scale transformation using logicle transformation [17] for fluorescent channels - with automated calibration using *flowCore* [18], and linear transformation for scattered light channels - aligning 5 and 95 quantiles of *FSC/SSC* channels to those of a reference fluorescent channel.

Subsequently, we performed visual QC, using *CytoMDS*, where we limited the sample distance calculation to the 4 preprocessing channels: FSC-A, FSC-H, SSC-A and Live/Dead. We identified 3 outlying samples, which we discarded after visual inspection of 2D plots of these suspicious samples (plots shown in Figure D.23). We also excluded the 16 *bridge* samples from model training, since the age of the bridge donor is unknown. Note we kept these *bridge* samples aside for external validation (see Results).

Then, we designed a *CytoCNN* architecture to predict donor age from the corresponding sample cytometry data. As the outcome is a continuous variable (regression problem), we applied a linear activation in the output layer, and used the MSE as a loss function (see Section 3.2.1). We also calculated the RMSE, the MAE, and the CCC as complementary evaluation metrics (see Section 2.6.2).

Here, since we had a reasonable sample size at our disposal, we chose to not use cross-validation, allowing us to avoid lengthy run times. Instead, we randomly selected a fixed validation set (20%, i.e. 96 samples) and a fixed testing set (another 20% of the samples). When doing this, we stratified by age group to obtain similar ratios of age groups in the training, validation and test sets. In line with Section 10.3, we systematically kept the best model seen during training, from the validation loss viewpoint, and took that particular model for testing and predictions.

Hyperparameter values were set according to the following choices:

- the input size was set to 50,000 randomly sub-sampled cells. For each initial random seed, a different sub-sample is taken from the original flow cytometry sample.
- the number of convolutional layers was set to 2, with 3 feature maps. The fully connected part of the network consisted of the merge layer with 3 nodes, one single intermediary dense layer with 3 neurons, and the output layer with 1 neuron. Indeed, we selected a minimum size network architecture with the aim to avoid overfitting. We did not add any additional patient specific covariate in the merge layer.

- the learning rate was tuned using grid search (see Section 2.5), and finally set to 0.001.
- the minibatch size was set to 20.

Using these hyperparameter choices, we defined a ‘base’ model for which we did not use any regularisation, dropout layers or data augmentation. In a second step, we applied L2 regularisation to stabilise the learning curves (see Results). We set the λ hyperparameter to 1, thanks to grid search.

Finally, an *ensemble* model was built by performing model averaging - a method also called *bagging* [25]. It consists in selecting a number of underlying models, and take as age prediction, the average of the individual model age predictions. The number of underlying models is a hyperparameter of the ensemble model, and has been set using grid search, i.e. by selecting the hyperparameter value corresponding to the best evaluation metrics in validation (see Results).

10.5.4 Results

Learning curves

Using the base model configuration described above, we trained 20 instances of *CytoCNN* model, for 20 different random seeds. Each training run, applied to 50,000 input cell samples, and with 10,000 epochs, took about 5 hours on a A100 GPU. Figure 10.16, subplots (a) to (d), shows the obtained learning curves, for one random seed. While the training curves seem fully satisfactory - loss function and RMSE decreasing smoothly and reaching a plateau at the end, the corresponding curves in validation are clearly unstable, having oscillations of huge amplitude. To mitigate this problem, we applied regularisation (see Methods), and obtained the plots in Figure 10.16, subplots (e) to (h). Here, the oscillatory behaviour is not completely solved, but at least the loss function presents a well visible ‘lower bound envelope’ - looking at the evolving minimum reached for each oscillation - which decreases smoothly and stabilises at the end. Figures D.24 to D.27, show that this behaviour is the same for each random seed - with notably two model instances - for random seeds 46 and 50 - having higher amplitude oscillations. Therefore, we decided to keep the regularised version of the model for the remainder of this section.

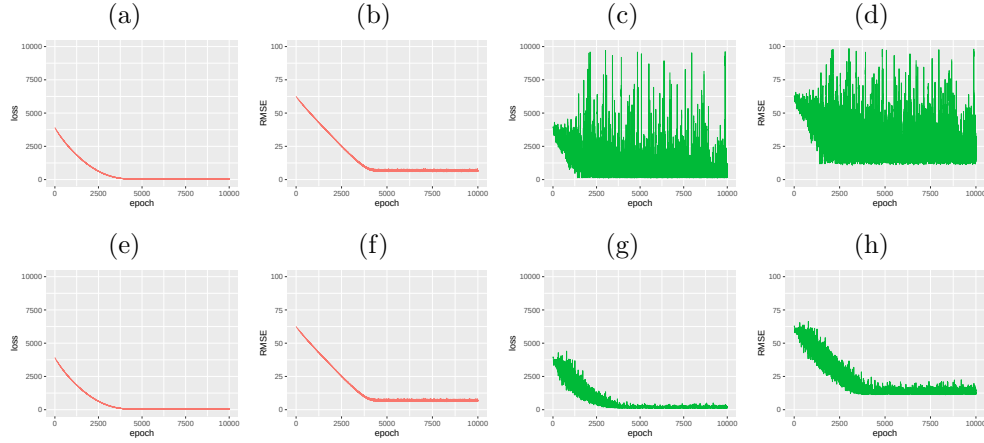


Figure 10.16: Learning curves obtained for random seed 42. In each row, we have, from left to right: the training loss, the training [RMSE](#), the validation loss, and the validation [RMSE](#). The first row - subplots (a) to (d) - corresponds to the base model; the second row - subplots (e) to (h) - corresponds to L2 regularisation.

Evaluation metrics and model predictions

Some model evaluation metric distributions are displayed in Figure 10.17, showing box plots in training, validation and test for the 20 model instances, both for the [RMSE](#) and [CCC](#). We obtain reasonably good [RMSE](#) in test - in the [8.86 (years) - 10.66 (years)] interval (observed range for the whole set of 20 models), and [CCC](#) in test - in the [0.70, 0.83] interval. Detailed metric results can be found in Table D.1, and scatter plots by random seed, in Figures D.28 to D.30.

Figure 10.18 depicts the 20 different model predictions for each subjects, by increasing true age. General trends are globally the same in validation (a) and in test (b):

- the dispersion of model predictions is higher for the group of younger individuals; this is most probably due to the relatively small amount of information available for this group - see Figure 10.15.
- looking specifically at the 50+ individuals, in general there is a tendency for the different models to overestimate the age of the youngest (mid age) individuals, and to underestimate the age of the oldest individuals. This is again probably due to the age distribution of the available data,

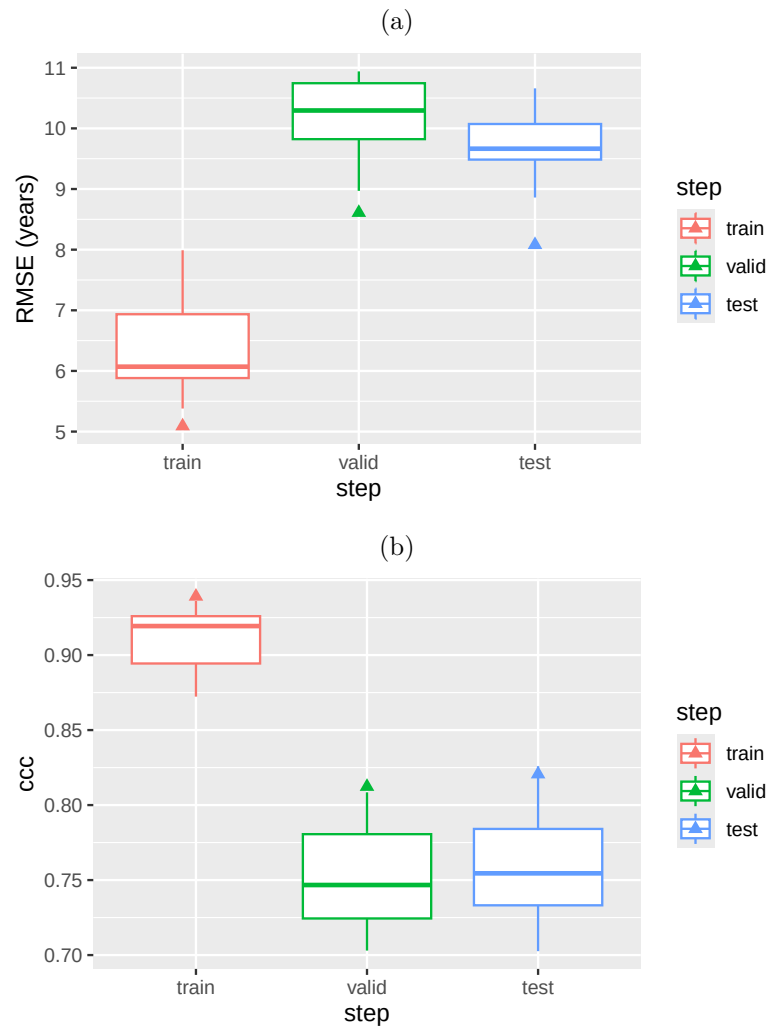


Figure 10.17: Distribution of some evaluation metrics obtained for the 20 different random seed models, in training, validation and test. (a) Box plots of [RMSE](#). (b) Box plots of [CCC](#). The triangles correspond to the metrics obtained with the ensemble model, with 15 underlying models.

Ensemble model

With the aim to further improve the evaluation metrics obtained in the previous paragraph, we took advantage of our 20 different model instances, and built ensemble models with various numbers of underlying models - see Methods. Figure 10.19 shows the RMSE obtained in validation, for an increasing number of underlying models. We found that the actual minimum was reached for a number of models $n = 15$ - although values between 10 and 20 lead to only slightly worse results. Taking $n = 15$, RMSE and CCC values in training, validation and test are shown - as little triangles - on Figure 10.17. Note that the ensemble model performs systematically better than any of the underlying models, which comforts our approach.

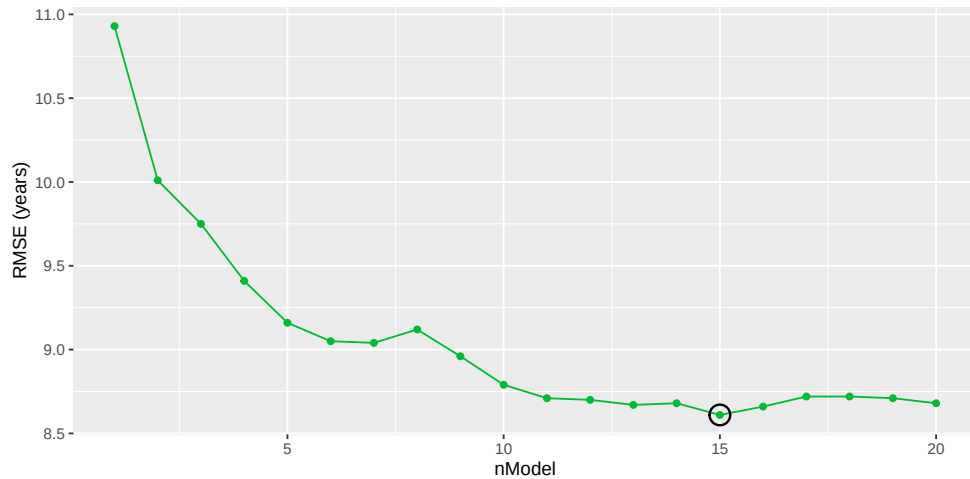


Figure 10.19: RMSE in validation, as a function of number of underlying models in the ensemble model. The best evaluation metric is attained for 15 models.

Scatter plots of individual age prediction, against true age, are represented in Figure 10.20. Similar plots, coloured by study or data acquisition runs, are also provided in Figures D.31 and D.32. In those additional plots, it seems that no particular patterns can be identified, which seems to be a good sign that the model predictions are not driven by potential confounders with age, i.e. study and run, as discussed in Section 10.5.2.



Figure 10.20: Ensemble model predicted age vs. true age, in years, by dataset.

Summary evaluation metric results are also provided in Table 10.1. Looking only at validation and test results, the CCC metric is globally around 0.80, which is reasonably good. However, if we recalculate it after splitting into the two age groups, it becomes much lower: around 0.50 for the 50+ individuals, and between 0 and 0.30 for younger individuals. It seems that the model can discriminate well between the two age groups, but is not able to identify smaller differences within age groups.

Metric	Training	Validation	Test
RMSE (years)	5.09	8.61	8.08
MAE (years)	4.10	6.78	6.53
CCC	0.94	0.81	0.82
CCC - older individuals	0.77	0.52	0.46
CCC - younger individuals	0.69	0.05	0.26

Table 10.1: Evaluation metrics obtained for the ensemble model, by dataset.

External validation using the *bridge* samples

With the aim to validate our ensemble model on other datasets, we calculated model predictions for the 16 *bridge* samples (see Section 10.5.2). Since these technical replicates originate from the same PBMC sample from the same donor, the predicted age for these should ideally be the same. Box plots of predicted ages, for the 20 model instances, as well as the ensemble model (red line), are shown in Figure 10.21. The predicted age is reasonably consistent between runs, generally slightly above 50 years old, at the exception of RUN5, for which the predictions are consistently lower - and maybe RUN17, for which the predictions are a bit more dispersed, with an ensemble model prediction

also a bit lower than for the other run samples. Investigating into RUN5, we could not identify a specific reason for this consistently lower predicted age. However, the results of this first external model validation exercise are rather satisfactory. Note that the true age of the bridge donor is unfortunately unknown - but as we were told, it should be less than 65 years old.

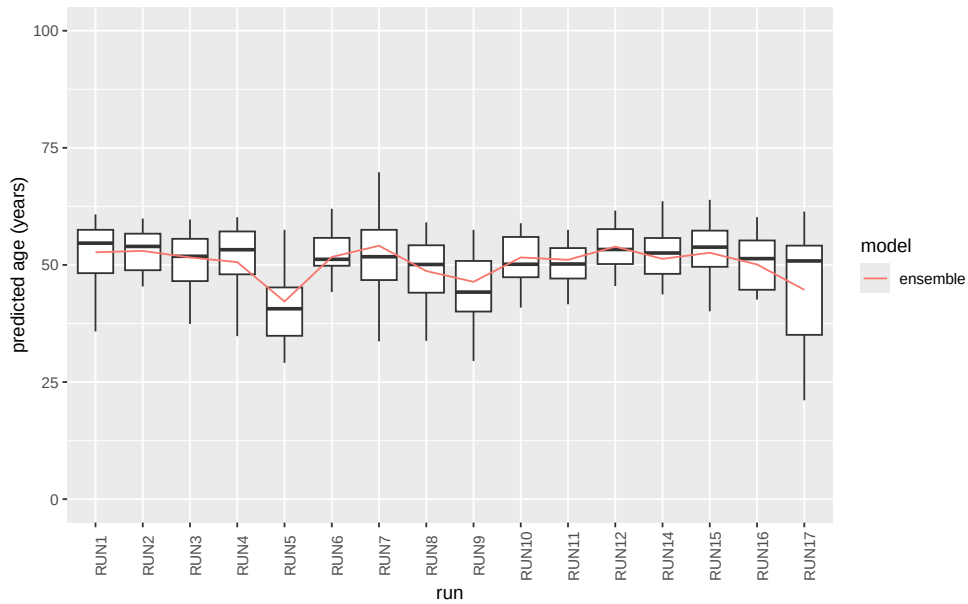


Figure 10.21: Box plots of age predictions for the *bridge* samples, in years, for the 16 data acquisition runs. The red line links the prediction values obtained by the ensemble model.

External validation using a panel testing dataset

A second external validation emanates from the fact that GSK scientists had performed preliminary data acquisition tests for the cytometry panel of markers under study, while they were fine-tuning it. As by-product of these tests, we obtained a ‘panel testing’ dataset, described elsewhere [10], and having the same set of markers as the main dataset. In short, this dataset contains 20 FCS files, corresponding to the flow cytometry data acquisition of 15 PBMC samples. These biological samples were collected and cryopreserved from healthy individuals, split into two age groups - younger and older adults - and were part of two different GSK biobanks. The flow cytometry data acquisition itself was performed in two different acquisition rounds (a former and a later data acquisition, done at 3 months time interval), and 5 of the

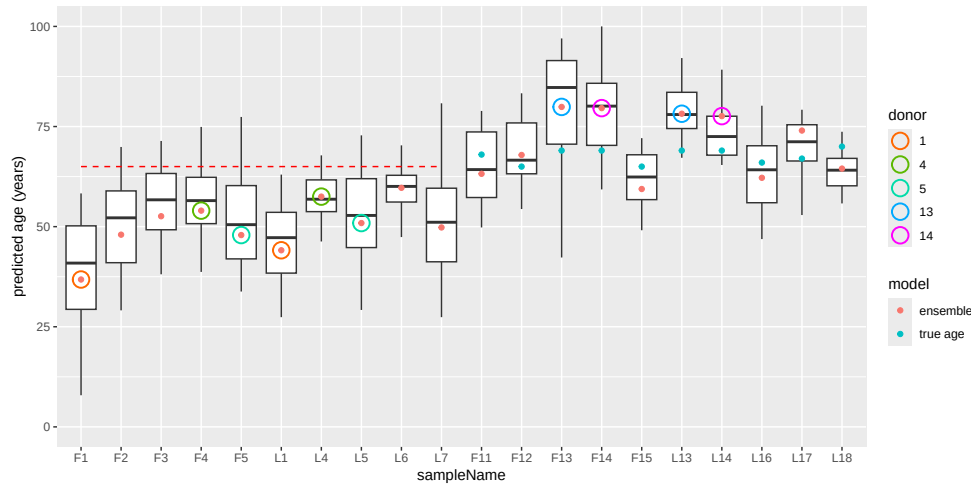


Figure 10.22: Box plots of age predictions, in years, for the panel testing samples. Samples F1 to L7, on the left half of the plot, correspond to younger adults, for which we do not know the exact ages, except that they lie under the 65 years old threshold (red dashed line). Samples F11 to L18, on the right half, correspond to older adults, for which we know the true age (blue dots). Inner red dots with same surrounding colour circles correspond to different data acquisitions of the same donor PBMC samples.

biological samples were acquired in both rounds.

Model prediction results for the ‘panel testing’ dataset are shown in Figure 10.22. On this plot, donors 1 to 7 are younger adults, while donors 11 to 18 are older adults. The true age of older adults is known, and shown as a blue dot on the plot, while we ignore the true age of younger adults - we only know with certainty that their age is below 65 years old. From the plot, we can conclude the following:

- younger adults always have a predicted age lower than 65 years old, and also lower than the predicted age for all older adults;
- older adults have predicted age that are ‘not too far’ from their true age, i.e. the prediction errors are compatible with the RMSE and MAE metrics reported above;
- when there exist technical replicates coming from two different acquisition runs (same colour circles on the plot), the predictions for these replicates are remarkably in line. This is not surprising since these sample pairs correspond to technical replicates. The proximity of these sample

pairs are also confirmed from the *CytoMDS* visualisation of the panel testing dataset - see Figure D.33.

All in all, we can say that these are good external validation results.

10.5.5 Discussion

The ensemble model is successful in discriminating between age groups, but lacks predictive granularity

In the reported results for the ensemble model, we obtained good model evaluation metrics in test - a [RMSE](#) of 8.1 years, a [MAE](#) of 6.5 years, and a [CCC](#) of above 0.8. Moreover, we could also validate our model predictions using two small external datasets. However, when looking into the details of model predictions, we could identify that the model, to a certain extent, seems to do a good job in distinguishing young adults from older adults, but is less able to detect subtle age differences within groups (Figure 10.18). Actually, based on our study, we are not able to find the root cause of this weakness. It could be the lack of available data, especially for the intermediate age and very old age ranges, or linked to decorrelation between chronological age and immune age, or something else, for example an intrinsic flaw of our model.

Model prediction errors cannot be explained by frailty index

One puzzling observation from the plots in Figure 10.18 is the fact that some individuals have consistently lower or higher age predictions than their true age. In the assumption that the model would do its intended job perfectly, this could be an indication of discrepancy between chronological age and immune age for these individuals. We tried to indirectly confirm this assumption by using additional data. Indeed, in one of the four vaccine studies at the origin of the *GSK Immunosenescence* dataset, the physical and mental state of elderly participants were systematically collected, and translated by medical practitioners into a capability measure called *frailty* index. This index can take any integer value between 1 (good health state and good overall capabilities) and 3 (bad health state and limited capabilities). Therefore we attempted to take advantage of these additional data, reasoning that a high frailty index could potentially be associated with high immune age. In that case, if the model prediction error would be driven by the discrepancy between immune age and chronological age, then a highly positive prediction error should be associated with a high frailty index, controlled for age.

We therefore extracted the model prediction errors for the 158 elderly for whom we had an available frailty index. A scatter plot of the data is shown



Figure 10.23: Scatter plot of the ensemble model prediction error vs. age (in year), with data points coloured according to the frailty index.

in Figure 10.23. We notice a positive association between high frailty index (blue and green dots) and age, which corresponds to the overall expectation. The plot also shows a clear negative correlation between our model prediction error and the true age of the individuals, and this pattern cannot be explained by the frailty index.

Then, we fitted the following linear regression model: $\text{prediction_error} \sim \text{scaled}(\text{age}) + \text{frailty_index}$ (Table 10.2). Controlled for the age variable, the estimator of the frailty index coefficient is negative (in the wrong direction, thus, with respect to our initial conjecture), but small and clearly not statistically significant. In contrast, age has a negative, highly significant contribution to the model response, i.e. our model prediction error. This means that what drives (systematic) model prediction error for a given individual, remains unexplained at this stage.

coef.	estimate	std-error	pvalue
intercept	0.41	1.17	7.3e-01
age	-0.45	0.05	6.7e-15
frailty index	-0.40	0.69	5.6e-01

Table 10.2: Inference table for the linear model: $\text{prediction_error} \sim \text{scaled}(\text{age}) + \text{frailty_index}$

Explaining the model is challenging

One issue with deep learning models is that they are ‘black box’ models, i.e. they are not readily interpretable. For example, with our ensemble model, we do not know why some individual data leads to a high or a low age prediction. However, this model interpretation is key to identify biomarkers of immune aging. As mentioned in Section 10.1, one of the appealing features of the results presented in [1] is that the authors, using their models, managed to identify a specific cell population of which the abundance could correlate with CMV status. To obtain this interpretation, the authors implemented the following XAI [6] procedure (Figure 10.24):

- Step 1: for each training sample i , for each cell j - corresponding to row j of the cytometry marker intensity matrix, create a new data matrix, by randomly replacing a number of other rows, e.g. 100 rows, by duplicates of the current cell row j .
- Step 2: submit the perturbed data matrix as input to the model, obtain a new model prediction, and calculate the difference with the original model prediction for the training sample i at hand. This difference, designated as ΔY in Figure 10.24, quantifies the influence of cell j in the model prediction of sample i .
- Step 3: gather the ΔY_{ij} for each cell j , and all training samples i , and build a regression tree model for predicting ΔY , using cytometry markers as regression variables.

Tree models have the advantage of being interpretable, in that they come with feature importance indicators for explaining high or low outcomes. In our case, this means that we would be able to find markers of which the intensity level would tend to explain the relatively high or low impact of a particular cell in the predicted age of an individual. In turn, this would enable us to identify typical combination of marker patterns - hence cell subpopulations - leading to high or low immune age.

We tried to implement the process described above with our *CytoCNN* models, for the prediction of chronological age, with the aim to make biological sense of our age predictions. However, our first tests were not directly conclusive, for a number of reasons. First, building a good regression tree presents in itself its own challenges, and some associated hyperparameters also need to be adequately chosen, for instance to avoid overfitting. Notably, choosing a target tree depth for our case was found to be challenging. Second, our ensemble model is itself based on a series of underlying model instances,

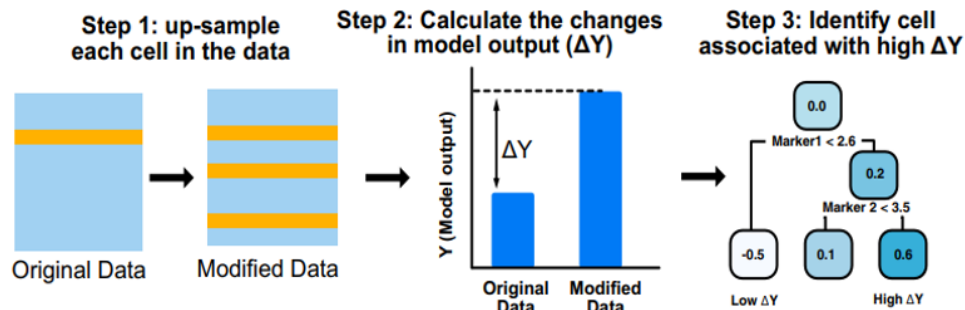


Figure 10.24: Schematic representation of the XAI technique applied in [1]. Figure adapted from [1].

each one having its own ΔY responses. Therefore, the need to combine regression trees for each underlying model adds another layer of complexity, given that each tree can flag different markers as important ones. As a consequence, developing XAI procedures for our model would still require additional development and testing work, which we have been unable to perform within the scope of this work.

Further extensions

Finally, we highlight a few possible extensions of this work that could serve as directions for future investigation:

- Given that our ensemble model effectively distinguishes between broad age groups (young vs. old) but shows limited accuracy within these groups - as discussed above - an interesting extension would be to apply a similar modelling strategy restricted to older adults (ages above 45 years). This could help elucidate the underlying reasons for the limited predictive granularity of the model.
- Rather than constructing regression trees as part of an XAI framework complementing our DL model, it would also be valuable to explore such a simpler ML approach directly. For example, one could derive a large set of features from automatically identified cell populations across all samples and use these as inputs to a regression tree model. This strategy would be conceptually similar to the *FloReMi.1* model [5] developed for the *FlowCapIV* challenge.

10.6 Overall conclusions

In this final section, we summarise the work done with *CytoCNN* and draw some general conclusions from our computational experiments with CNN models on cytometry data.

As a first proof of concept, in Section 10.3, we tested our *CytoCNN* framework for classifying the CMV status of healthy individuals, based on mass cytometry data, a task presented in [1], which was the inspiration for this work. We managed to obtain a similar model predictive performance as reported in the article, which gave us confidence in our implementation, and supported the idea of testing the framework in further use cases.

In Section 10.4, we fine-tuned the *CytoCNN* components to handle a survival outcome, and trained various model instances on the *FlowCapIV* public dataset, aiming at predicting the time to progression to AIDS for HIV seropositive patients. Unfortunately, we did not succeed in that task, due to a major overfitting problem, which could not be resolved by standard techniques often used to ‘fight’ overfitting in deep neural networks.

Finally, in Section 10.5, we used our *CytoCNN* workhorse on a regression problem, trying to predict the chronological age of individuals - as a proxy for immune age - using flow cytometry data collected in the *GSK Immunosenescence* dataset. There, we managed to get consistent and reproducible results from run to run. Also, using an ensemble model approach, averaging over several individual *CytoCNN* model instances, we succeeded in obtaining decent evaluation metrics in validation and test. We also obtained reasonably good external validation results on two small sample sets. However, we could not verify whether, for model fitting, chronological age was or not a good proxy for immune age. More importantly, we could not (yet) explain *why* the final ensemble model predicts low or high ages, and therefore, we did not fulfill the important task of highlighting biomarkers of immune age in flow cytometry data.

There are many possible extensions to this work, which could still be explored. The most important one would probably be to continue the work on XAI techniques, with as goal to shed light on the ‘black box’ model and possibly make biological sense of the obtained model predictions. Then, there would be a possibility to further exploit the *GSK Immunosenescence* dataset, in order to try to predict vaccine responsiveness, and possibly identify cytom-

etry bio-markers of good or weak vaccine response, possibly linked to immune age. In our endeavours, we had run some preliminary tests of using *CytoCNN* for vaccine response classification, using a single study of the dataset, but we were quickly faced with similar overfitting problems as in Section 10.4, which again could be explained by insufficient training data available in one single study. One possible route of investigation could be to use the models built for age prediction (Section 10.5), and apply transfer learning techniques [26], to re-use the identified age-related features extracted from cytometry data, to predict vaccine response for a given study.

Another possible research avenue would be to try out amending the network input, by replacing cytometry data matrices by a series of 2D histogram representations, similar to typical plots used by cytometry experts to manually gate cytometry samples. This would then require the modification of the convolutional layers of the network to make them more in line with those used in computational vision tasks.

Along the journey of designing and testing deep neural networks for predicting various outcomes, using cytometry data as input, we faced a number of issues. First, our models needed *sufficient* training data, otherwise massive overfitting occurred. What is actually meant by ‘sufficient data’ depends heavily on the information contained in each training sample, in relation with the prediction task at hand. For instance, in Section 10.4, we were misled by a dataset containing hundreds of samples, but with a high proportion of early censored data, having low useful information content. Second, the development and fine-tuning of these models is complicated. They typically contain many hyperparameters, for which ‘good’ values need to be identified, requiring considerable amount of trial and error. Third, these models are computationally extensive and their training require heavy, economically and environmentally costly, dedicated IT infrastructures. Fourth, they are complex, meaning that well designed diagnostic tools, like exploratory data analysis plots, or plots like the ones introduced in Section 10.4.3.1, are paramount for understanding what is happening. Fifth, because of the complexity, it is easy to make mistakes, and notably to fall into the *data leakage* trap, as experienced in Section 10.4.3.2. And, last but not least, these models are not readily interpretable, meaning that an extra layer of XAI techniques is needed for extracting the essence of these models. Therefore, a key take home message is that, all in all, and despite their ubiquity, popularity, and success in many prediction tasks, using deep learning models for medical research still requires a great deal of awareness, persistence and wisdom.

References for Chapter 10

- [1] Zicheng Hu et al. “A robust and interpretable end-to-end deep learning model for cytometry data”. en. In: *Proc. Natl. Acad. Sci. U. S. A.* 117.35 (Sept. 2020), pp. 21373–21380. DOI: [10.1073/pnas.2003026117](https://doi.org/10.1073/pnas.2003026117).
- [2] Eirini Arvaniti and Manfred Claassen. “Sensitive detection of rare disease-associated cell subsets via representation learning”. en. In: *Nat. Commun.* 8 (Apr. 2017), p. 14825. DOI: [10.1038/ncomms14825](https://doi.org/10.1038/ncomms14825).
- [3] Sanchita Bhattacharya et al. “ImmPort, toward repurposing of open access immunological assay data for translational and clinical research”. en. In: *Sci. Data* 5 (Feb. 2018), p. 180015. DOI: [10.1038/sdata.2018.15](https://doi.org/10.1038/sdata.2018.15).
- [4] Sofie Van Gassen et al. “FlowSOM: Using self-organizing maps for visualization and interpretation of cytometry data”. en. In: *Cytometry A* 87.7 (July 2015), pp. 636–645. DOI: [10.1002/cyto.a.22625](https://doi.org/10.1002/cyto.a.22625).
- [5] Sofie Van Gassen et al. “FloReMi: Flow density survival regression using minimal feature redundancy”. en. In: *Cytometry A* 89.1 (Jan. 2016), pp. 22–29. DOI: [10.1002/cyto.a.22734](https://doi.org/10.1002/cyto.a.22734).
- [6] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why should I trust you?: Explaining the predictions of any classifier”. en. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, Aug. 2016. DOI: [10.1145/2939672.2939778](https://doi.org/10.1145/2939672.2939778).
- [7] Charles R Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *arXiv [cs.CV]* (Dec. 2016). URL: <http://arxiv.org/abs/1612.00593>.
- [8] Badr-Ali Mouaden, Laurent Gatto, and Michel Verleysen. “Applying Deep Convolution Neural Network (CNN) on flow cytometry data for the prediction of a survival type clinical outcome”. MA thesis. Ecole polytechnique de Louvain, Université catholique de Louvain, 2023. URL: <https://dial.uclouvain.be/memoire/ucl/object/thesis:43248>.

- [9] Simon Wiegerebe et al. “Deep learning for survival analysis: a review”. en. In: *Artif. Intell. Rev.* 57.3 (Feb. 2024), pp. 1–34. DOI: [10.1007/s10462-023-10681-3](https://doi.org/10.1007/s10462-023-10681-3).
- [10] Philippe Hauchamps et al. “Visual Quality Control With CytoMDS, a Bioconductor Package for Low Dimensional Representation of Cytometry Sample Distances”. In: *Cytometry Part A* 107.3 (2025), pp. 177–186. DOI: [10.1002/cyto.a.24921](https://doi.org/10.1002/cyto.a.24921).
- [11] Nima Aghaeepour et al. “Critical assessment of automated flow cytometry data analysis techniques”. en. In: *Nat. Methods* 10.3 (Mar. 2013), pp. 228–238. DOI: [10.1038/nmeth.2365](https://doi.org/10.1038/nmeth.2365).
- [12] Nima Aghaeepour et al. “A benchmark for evaluation of algorithms for identification of cellular correlates of clinical outcomes”. en. In: *Cytometry A* 89.1 (Jan. 2016), pp. 16–21. DOI: [10.1002/cyto.a.22732](https://doi.org/10.1002/cyto.a.22732).
- [13] Josef Spidlen et al. “FlowRepository: a resource of annotated flow cytometry datasets associated with peer-reviewed publications”. en. In: *Cytometry A* 81.9 (Sept. 2012), pp. 727–731. DOI: [10.1002/cyto.a.22106](https://doi.org/10.1002/cyto.a.22106).
- [14] Philippe Hauchamps et al. “CytoPipeline and CytoPipelineGUI: a Bioconductor R package suite for building and visualizing automated pre-processing pipelines for flow cytometry data”. In: *BMC Bioinformatics* 25.1 (Feb. 2024), p. 80. DOI: [10.1186/s12859-024-05691-z](https://doi.org/10.1186/s12859-024-05691-z).
- [15] Annelies Emmaneel et al. “PeacoQC: Peak-based selection of high quality cytometry data”. en. In: *Cytometry A* 101.4 (Apr. 2022), pp. 325–338. DOI: [10.1002/cyto.a.24501](https://doi.org/10.1002/cyto.a.24501).
- [16] Florian Hahne et al. “Per-channel basis normalization methods for flow cytometry data”. en. In: *Cytometry A* 77.2 (Feb. 2010), pp. 121–131. DOI: [10.1002/cyto.a.20823](https://doi.org/10.1002/cyto.a.20823).
- [17] David R Parks, Mario Roederer, and Wayne A Moore. “A new “Logicle” display method avoids deceptive effects of logarithmic scaling for low signals and compensated data”. en. In: *Cytometry A* 69.6 (June 2006), pp. 541–551. DOI: [10.1002/cyto.a.20258](https://doi.org/10.1002/cyto.a.20258).
- [18] B Ellis et al. *flowCore: flowCore: Basic structures for flow cytometry data*. R package version 2.21.0. 2025. DOI: [10.18129/B9.bioc.flowCore](https://doi.org/10.18129/B9.bioc.flowCore). URL: <https://bioconductor.org/packages/flowCore>.
- [19] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *J. Mach. Learn. Res.* 15.56 (2014), pp. 1929–1958. DOI: [10.5555/2627435.2670313](https://doi.org/10.5555/2627435.2670313).

- [20] Amir Ajoolabady et al. “Immunosenescence and inflammaging: Mechanisms and role in diseases”. en. In: *Ageing Res. Rev.* 101.102540 (Nov. 2024), p. 102540. DOI: [10.1016/j.arr.2024.102540](https://doi.org/10.1016/j.arr.2024.102540).
- [21] Annalisa Ciabattini et al. “Vaccination in the elderly: The challenge of immune changes with aging”. en. In: *Semin. Immunol.* 40 (Dec. 2018), pp. 83–94. DOI: [10.1016/j.smim.2018.10.010](https://doi.org/10.1016/j.smim.2018.10.010).
- [22] Tingmei Zhao et al. “Vaccine adjuvants: mechanisms and platforms”. en. In: *Signal Transduct. Target. Ther.* 8.1 (July 2023), p. 283. DOI: [10.1038/s41392-023-01557-7](https://doi.org/10.1038/s41392-023-01557-7).
- [23] Gilles Berrut and Laure de Decker. “[Immunosenescence: a review]”. fr. In: *Geriatr. Psychol. Neuropsychiatr. Vieil.* 13 Suppl 2 (Sept. 2015), pp. 7–14. DOI: [10.1684/pnv.2015.0548](https://doi.org/10.1684/pnv.2015.0548).
- [24] Greg Finak, Wenxin Jiang, and Raphael Gottardo. “CytoML for cross-platform cytometry data sharing”. en. In: *Cytometry A* 93.12 (Dec. 2018), pp. 1189–1196. DOI: [10.1002/cyto.a.23663](https://doi.org/10.1002/cyto.a.23663).
- [25] Peter Bühlmann. “Bagging, Boosting and Ensemble Methods”. In: *Handbook of Computational Statistics: Concepts and Methods*. Ed. by James E. Gentle, Wolfgang Karl Härdle, and Yuichi Mori. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 985–1022. ISBN: 978-3-642-21551-3. DOI: [10.1007/978-3-642-21551-3_33](https://doi.org/10.1007/978-3-642-21551-3_33).
- [26] Asmaul Hosna et al. “Transfer learning: a friendly introduction”. en. In: *J. Big Data* 9.1 (Oct. 2022), p. 102. DOI: [10.1186/s40537-022-00652-w](https://doi.org/10.1186/s40537-022-00652-w).

Part III

Discussion

Chapter 11

Summary, conclusion and perspectives

11.1 Summary of scientific contributions

In this section, we provide a broader overview of the key scientific contributions of our work, beyond the scientific articles presented in the preceding chapters. These contributions fall into two main categories. The first concerns the identification of key challenges and risks associated with automated data analysis in general, and cytometry data in particular - issues that may be less pronounced or absent in manual approaches. The second consists of a suite of tools and software packages developed during the course of this project to address these challenges. These deliverables represent practical outcomes derived from opportunities identified throughout our research.

11.1.1 Challenges and risks of automated cytometry data analyses

Challenge 1: automated data preprocessing pipelines are complex to design, difficult to evaluate, and challenging to maintain.

As discussed in Chapter 4, automated cytometry data analysis typically begins with a multi-step preprocessing pipeline. However, as highlighted in Chapter 7, such pipelines pose several significant challenges to the bioinformatician.

First, constructing a robust and fit-for-purpose preprocessing pipeline requires a series of informed decisions: selecting appropriate methods for each

step from a wide array of alternatives, determining suitable parameter values, and defining an execution order that avoids harmful interactions between steps. This *combinatorial problem of designing pipelines* (see Chapter 7) makes it highly impractical to exhaustively and systematically evaluate a representative set of pipeline configurations. Nevertheless, such evaluation is critical, as some methods may lack robustness and fail to yield satisfactory results in edge cases.

Second, the evaluation of preprocessing pipelines is further complicated by the absence of standardised diagnostics and visualisations. The benchmarking tools and studies currently available often provide limited insight into pipeline performance, typically focusing on summary metrics without offering support for deeper investigation. This lack of interpretability makes it difficult for users to understand *why* a given pipeline performs better than another, hindering both performance optimisation and methodological transparency.

Third, integrating multiple preprocessing steps into a workflow frequently results in software that is difficult to understand, maintain, and extend. This problem is compounded by the evolving nature of software packages that implement the underlying methods, where frequent updates may introduce changes without clear documentation of their impact on pipeline behaviour.

These challenges introduce critical risks. The main one is probably the temptation to rely on generic, ‘one-size-fits-all’ preprocessing pipelines that are applied indiscriminately across datasets and research questions, without critical evaluation of each step or its interactions with others, as well as the impact on downstream analysis results. Another related risk is the uncritical upgrading of pipeline components following upstream software updates, which can introduce regressions that go unnoticed in the absence of rigorous validation. However, this latter risk can be mitigated by systematic software regression testing during pipeline maintenance and updates - though such measures require dedicated expertise and resources.

Challenge 2: rigorous QC of acquired samples is laborious but critical for reliable automated analyses.

As discussed in Section 4.3, several automated tools are available to perform key preprocessing tasks aimed at preventing low-quality data from contaminating downstream analyses. However, as already mentioned in Chapter

8, no matter how advanced or automated these tools are, some level of manual QC remains indispensable. In practice, this often involves visual inspection of individual samples at various stages of preprocessing, typically performed by human operators using GUI-equipped softwares. For large scale studies involving hundreds of samples, this process can quickly become time-consuming, labour-intensive, and difficult to manage.

Despite its complexity, rigorous QC is paramount when relying on automated analysis pipelines. Multiple studies have underscored the detrimental impact of poor-quality data on downstream results. For example, Emmaneel et al. [1] demonstrated how technical artefacts such as signal instability can distort cell population clustering, while Mazza et al. [2] showed that background fluorescence and spreading error can significantly affect data visualisation using tSNE.

It is also important to note that automated algorithms are, in some respects, more sensitive to data quality issues than manual analysis. Interactive manual gating allows scientists to identify and correct for quality problems in real time - e.g. by adjusting gates to compensate for batch effects, by excluding aberrant events, or by ignoring problematic markers. Such flexible, expert-driven interventions are generally not available in automated workflows.

Consequently, without a comprehensive and rigorous QC process, automated analyses are more susceptible to undetected quality issues, which can propagate through the pipeline and ultimately compromise the validity of the results ('garbage in, garbage out').

Challenge 3: interacting transparently with automated cytometry data analysis pipelines can be challenging for many scientists.

Automated cytometry data analysis pipelines are complex, comprising numerous interdependent steps. On the one hand, some of these steps could be available as integrated functionality into turnkey software platforms such as *FlowJo* (see Section 5.2), which streamline user interaction and reduce the technical barrier. However, this convenience often comes at the cost of reduced transparency, limited control over key parameters, and restricted opportunities for critical assessment of the underlying methods. On the other hand, many advanced or emerging methods are not yet supported by such platforms and instead require direct interaction through scripting environments like R

or Python. This presents a significant challenge for practitioners without a strong computational background. In fact, there is a lack of well-designed interactive tools that bridge this gap - tools that would allow users to engage more intuitively with parameter tuning and method interpretation, while still leveraging the power of advanced algorithms.

As a consequence, users are often left with two problematic options: either to rely uncritically on black-box implementations within point-and-click software or fully automated pipelines, or to overlook important developments in methods implementing key steps of data analysis. Both options risk compromising the integrity and reliability of the downstream analysis, as previously discussed.

Challenge 4: DL methods are powerful and popular, but their responsible use requires substantial effort, awareness, and critical thinking.

When the objective of a cytometry data analysis pipeline is to predict clinical outcomes - such as in biomarker discovery - DL methods offer an attractive solution. These models have the potential to capture complex, non-linear relationships between cytometry data and clinical endpoints. In some cases, they can even partially circumvent traditional preprocessing steps, such as batch effect correction (see Section 10.1).

In recent years, DL has become ubiquitous in healthcare applications [3, 4, 5, 6, 7], and the tools supporting it have significantly improved in terms of usability and accessibility. Today, building and deploying DL models no longer requires programming expertise. For example, a variety of commercial platforms - such as Google Cloud AutoML, Microsoft Azure ML Designer, and Amazon SageMaker Canvas - even offer end-to-end, turnkey solutions for the entire ML workflow. These platforms allow users to upload data, select models, assess performance, and deploy results through user-friendly graphical interfaces. However, this perceived ease of use can be misleading. As discussed throughout Chapter 10, applying these methods without a critical understanding of their limitations entails serious risks.

One major challenge is the need for careful model tuning - particularly through hyperparameter optimisation - which is often labourious and time-consuming. Another difficulty lies in determining how much labelled training

data is required, a factor that depends on both the structure and complexity of the dataset and the strength of the underlying biological signal. Perhaps the most fundamental challenge is the inherent ‘black-box’ nature of [DL](#) models. This lack of interpretability complicates biological insight, with a need to develop sophisticated, often problem-specific, explainability methods. Finally, it is important to remember that even advanced models, such as deep learning models, cannot extract information that is not present in the data, nor can they compensate for poor data quality - the principle of ‘garbage in, garbage out’ still applies.

A key risk is that inexperienced users may adopt [DL](#) techniques without fully understanding these challenges, potentially leading to unreliable or misleading results. Additionally, the computational burden for training such models is significant, with notable environmental costs due to the use of [HPC](#) infrastructures.

11.1.2 Tools and software contributions

To address the challenges outlined above and mitigate associated risks, we developed a set of software tools, primarily implemented as R packages. These packages adhere to the coding and documentation standards of the Bioconductor project, supporting high-quality, open, and reproducible research.

The *CytoPipeline* suite of packages

To address Challenge 1, we developed the *CytoPipeline* R package (see Chapter 7). This tool is intended for bioinformaticians seeking to design, evaluate, and compare cytometry data preprocessing pipelines. It facilitates the systematic assessment of alternative pipeline configurations and provides in-depth insight into pipeline performance. *CytoPipeline* was first released as part of Bioconductor version 3.17 in April 2023.

Shortly thereafter, we introduced a companion package, *CytoPipelineGUI*, focusing on interactive visualisation. It includes two Shiny applications that assist in evaluating the performance of preprocessing steps. Unlike *CytoPipeline*, which is primarily aimed at bioinformaticians, *CytoPipelineGUI* also caters to end-users of preprocessing pipelines - such as experimented scientists - who may prefer a graphical interface over scripting. In this way, *CytoPipelineGUI* also addresses Challenge 3. This package was released in Bioconductor version

3.18 in October 2023.

Together, the *CytoPipeline* suite provides a flexible and extensible framework for building, executing, and visually inspecting preprocessing workflows. As demonstrated in Chapter 7, these tools enhance productivity during pipeline development and testing. They also complement benchmarking tools and studies by offering intuitive insights into performance outcomes.

We placed strong emphasis on comprehensive documentation, delivered through package vignettes. Additionally, a series of tutorial videos was created and made available via YouTube, linked from the vignettes. These resources offer a clear and accessible introduction to the concepts and functionality of the tools.

CytoMDS* and *MDSvis

To contribute to a more comprehensive QC process - specifically addressing Challenge 2 - we focused on developing a low dimensional summary visualisation that displays all samples in a study simultaneously, with interpretable distances, to facilitate the identification of data quality issues. This form of visual inspection serves as an additional QC step, complementing conventional sample-by-sample ‘QC in time’ approaches.

With this goal in mind, we developed the *CytoMDS* R package (see Chapter 8). This package implements an MDS projection based on an input matrix of EMDs between samples. *CytoMDS* was released in Bioconductor version 3.19 in April 2024.

Beyond the core low dimensional projection functionality, *CytoMDS* offers two key areas of added value:

- It is technically optimised for performance, with efficient memory handling and support for multicore processing. This enables the visualisation of large-scale study datasets within reasonable computational times.
- It provides advanced diagnostic tools for evaluating projection quality, as well as interpretability features to help users understand what drives variation along the projection axes. The latter feature is essential for linking visual patterns to sample characteristics, hence for identifying underlying quality issues.

In Chapter 8, we demonstrated - across three datasets and use cases - that *CytoMDS* enables rapid and global detection of data quality problems. Moreover, it can be applied at different stages of the analysis pipeline, where it may reveal issues resulting from automated processing steps.

To further improve accessibility - and again address Challenge 3 - we developed an additional R package, *MDSvis*, that provides an interactive Shiny application for exploring *CytoMDS* projections. Recognising that metric MDS (Section 4.4.4) has broader applications beyond cytometry data, we designed the software to support a generalised workflow, making it suitable for diverse data types and use cases (Chapter 9). *MDSvis* and its accompanying documentation are currently available via the following GitHub repository: <https://github.com/UCLouvain-CBIO/MDSvis>.

The *CytoCNN* framework and accompanying report

In Chapter 10, motivated by the growing interest in applying DL models to predict clinical outcomes - and to potentially identify associated biomarkers within patient cytometry data, we developed a deep CNN framework tailored for this purpose. Implemented in Python using the keras/TensorFlow libraries, this framework was named *CytoCNN*. We tested it across three different real-world use cases, achieving varying levels of success.

Importantly, the process of designing and evaluating *CytoCNN* led us to a deeper understanding of Challenge 4. In addition to the framework itself, we believe that the comprehensive report provided in Chapter 10 is a scientific contribution in itself, serving a broader purpose: raising awareness of the potential pitfalls and risks associated with the use of DL methods in biomedical research, when not applied with sufficient care and critical reflection.

11.2 Conclusion and perspectives

Over the past two decades, automated cytometry data analysis has gained prominence as a necessary complement to manual gating techniques. This shift has been largely driven by the rise of high dimensional cytometry technologies, with experimental panels including 30 to 50 markers. In such contexts, relying solely on traditional manual approaches has become increasingly impractical, highlighting the need for more scalable, reproducible, and data-driven solutions.

In this work, we have explored various steps of automated cytometry data analysis pipelines through a combination of literature review and hands-on experimentation. Through this process, we identified a series of challenges and risks that are either unique to or more pronounced in automated, algorithm-driven approaches compared to their manual counterparts. These issues are summarised in Section 11.1.1.

At the same time, our critical evaluation of existing workflows revealed several opportunities to design novel computational tools tailored to address some of these specific challenges. The resulting software contributions have been presented and discussed in details in Chapters 7 to 10, and are summarised in Section 11.1.2.

With the exception of the *CytoCNN* framework - which was developed in Python - all of these tools were implemented as R packages and released through the Bioconductor project. In doing so, we believe that this work actively contributes to the principles of open science and promotes reproducible research in the field of cytometry data analysis.

In terms of future research and potential directions for deepening this work, several avenues can be envisioned.

First, with regard to the preprocessing and QC tools - namely *CytoPipeline* and *CytoMDS*, along with their respective interactive Shiny applications, *CytoPipelineGUI* and *MDSvis* - there are multiple opportunities for further enhancement, as already outlined in the conclusion sections of their corresponding chapters. One particularly appealing possibility would be to integrate or interlink the two Shiny applications, thereby leveraging on their complementary functionalities. For example, a user could begin with a global QC overview of all samples using *MDSvis* (based on *CytoMDS*) to identify potentially problematic samples, and then seamlessly drill down into those specific cases using *CytoPipelineGUI* for a detailed inspection of the corresponding marker intensity distributions. Conversely, starting from the *CytoPipeline* interface, the user could iteratively invoke *CytoMDS* projections to assess the effect of each preprocessing stage on a global view of sample distances, thereby identifying potential robustness issues in the applied methods. Another valuable extension to *CytoPipeline* would be to broaden its current scope - currently limited to preprocessing - by including downstream analysis tasks such as batch effect correction and cell population identification.

Second, from a broader project perspective, the work initiated here could be extended to focus more specifically on the later stages of automated cytometry data analysis pipelines - particularly population identification and statistical analysis, as outlined in Section 4.1. This would involve a systematic review and hands-on exploration of a wide range of methods and algorithms, again with a critical eye toward identifying specific challenges and risks, as well as uncovering new opportunities for methodological innovation.

Finally, as discussed in Chapter 10, the *CytoCNN* framework itself presents numerous avenues for further development and experimentation. These could include the design of tailored explainable XAI techniques to interpret model predictions in biologically meaningful ways, as well as the application of the framework to new clinical endpoints, such as vaccine response. Indeed, the use of DL models in cytometry data analysis opens a wide array of possible directions. However, it is essential to emphasize that such explorations must be pursued with methodological rigor, a clear understanding of the limitations of DL models, careful validation of their outputs, and a commitment to their responsible use in biomedical research.

References for the Discussion part

- [1] Annelies Emmaneel et al. “PeacoQC: Peak-Based Selection of High Quality Cytometry Data”. en. In: *Cytometry A* (Sept. 2021). DOI: [10.1002/cyto.a.24501](https://doi.org/10.1002/cyto.a.24501).
- [2] Emilia Maria Cristina Mazza et al. “Background fluorescence and spreading error are major contributors of variability in high-dimensional flow cytometry data visualization by t-distributed stochastic neighboring embedding”. en. In: *Cytometry A* 93.8 (Aug. 2018), pp. 785–792.
- [3] Nurul Athirah Nasarudin et al. “A review of deep learning models and online healthcare databases for electronic health records and their use for health prediction”. en. In: *Artif. Intell. Rev.* 57.9 (Aug. 2024), pp. 1–26. DOI: [10.1007/s10462-024-10876-2](https://doi.org/10.1007/s10462-024-10876-2).
- [4] Chunjie Zhou et al. “A comprehensive review of deep learning-based models for heart disease prediction”. en. In: *Artif. Intell. Rev.* 57.9 (Aug. 2024), pp. 1–50. DOI: [10.1007/s10462-024-10899-9](https://doi.org/10.1007/s10462-024-10899-9).
- [5] Zarana Gajjar. “Deep learning approaches in healthcare: A comprehensive review of techniques and applications”. en. In: *J. Inf. Syst. Eng. Manag.* 10.55s (June 2025), pp. 01–12. ISSN: 2468-2071.
- [6] Hossein Sadr et al. “Unveiling the potential of artificial intelligence in revolutionizing disease diagnosis and prediction: a comprehensive review of machine learning and deep learning approaches”. en. In: *Eur. J. Med. Res.* 30.1 (May 2025), p. 418. DOI: [10.1186/s40001-025-02680-7](https://doi.org/10.1186/s40001-025-02680-7).
- [7] Riccardo Miotto et al. “Deep learning for healthcare: review, opportunities and challenges”. en. In: *Brief. Bioinform.* 19.6 (Nov. 2018), pp. 1236–1246. DOI: [10.1093/bib/bbx044](https://doi.org/10.1093/bib/bbx044).

Part IV

Appendices

Appendix A

Concordance Correlation Coefficient (CCC)

In this appendix, we derive the formula for the CCC, and we contrast it with the Pearson's correlation. The main content of this section is adapted from [1].

A.1 The concordance correlation coefficient for a bivariate distribution

Let us assume that pairs of samples $(Y_{i1}, Y_{i2}), i = 1, 2, \dots, n$, are independently selected from a bivariate population with means μ_1 and μ_2 and covariance matrix

$$\begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}.$$

The degree of concordance between Y_1 and Y_2 can be characterized by the expected value of the squared difference:

$$\begin{aligned} E[(Y_1 - Y_2)^2] &= (\mu_1 - \mu_2)^2 + (\sigma_1^2 + \sigma_2^2 - \sigma_{12}) \\ &= (\mu_1 - \mu_2)^2 + (\sigma_1 - \sigma_2)^2 + 2(1 - \rho)\sigma_1\sigma_2 \end{aligned}$$

, where ρ is the Pearson correlation coefficient. This also represents the expected squared perpendicular deviation from the 45 degree line, multiplied by 2. If each pair, Y_1 , and Y_2 , in the population are in perfect agreement, $E[(Y_1 - Y_2)^2]$ would be 0.

The following transformation is applied in order for the value of the coefficient to be scaled between -1 and 1 :

$$\rho_c = 1 - \frac{E[(Y_1 - Y_2)^2]}{\sigma_1^2 + \sigma_2^2 + (\mu_1 - \mu_2)^2}$$

or,

$$\rho_c = \frac{2\sigma_{12}}{\sigma_1^2 + \sigma_2^2 + (\mu_1 - \mu_2)^2} = \rho C_b, \quad (\text{A.1})$$

where

$$\begin{aligned} C_b &= [(v + 1/v + u^2)/2]^{-1}, \\ v &= \sigma_1/\sigma_2 = \text{scale shift}, \\ u &= (\mu_1 - \mu_2)/\sqrt{\sigma_1\sigma_2} = \text{location shift relative to the scale}. \end{aligned}$$

Here, $0 < C_b < 1$ is a bias correction factor that measures how far the best-fit line deviates from the 45 degree line (measure of accuracy). No deviation from the 45 degree line occurs when $C_b = 1$. The further C_b is from 1, the greater the deviation is from the 45 degree line. The Pearson correlation coefficient ρ measures how far each observation deviates from the best-fit line (measure of precision).

A.2 Formulas for n independent pairs of samples

When estimating the CCC, based on n independent pairs of samples, one need to use the ‘sample counterpart’ of A.1:

$$\text{CCC} = \frac{2S_{12}}{S_1^2 + S_2^2 + (\bar{Y}_1 - \bar{Y}_2)^2} \quad (\text{A.2})$$

where

$$\bar{Y}_j = \frac{1}{n} \sum_{i=1}^n Y_{ij}, \quad S_j^2 = \frac{1}{n} \sum_{i=1}^n (Y_{ij} - \bar{Y}_j)^2, \quad j = 1, 2;$$

and

$$S_{12} = \frac{1}{n} \sum_{i=1}^n (Y_{i1} - \bar{Y}_1)(Y_{i2} - \bar{Y}_2).$$

In contrast to Equation A.2, the Pearson correlation is estimated using the following formula:

$$\hat{\rho} = \frac{S_{12}}{S_1 S_2} \tag{A.3}$$

Reference

- [1] L I Lin. “A concordance correlation coefficient to evaluate reproducibility”. en. In: *Biometrics* 45.1 (Mar. 1989), pp. 255–268. DOI: [10.2307/2532051](https://doi.org/10.2307/2532051).

Appendix B

Supplementary results for Chapter 7

Supplementary Materials

Supplementary Tables

Step	Name	R package	Version	Method
1	flowframe_read	flowCore	2.12.0	read.flowSet()
2	remove_margins	PeacoQC	1.10.0	removeMargins()
3	compensate	flowCore	2.12.0	compensate()
4	flowframe_aggregate	CytoPipeline	1.0.0	aggregateAndSample()
5	scale_transform_estimate	flowCore	2.12.0	estimateLogicle()

Table S1. Scale transformation steps, common to both PeacoQC-based and flowAI-based pipelines.

Step	Name	R package	Version	Method
1	flowframe_read	flowCore	2.12.0	read.FCS()
2	remove_margins	PeacoQC	1.10.0	removeMargins()
3	compensate	flowCore	2.12.0	compensate()
4	perform_QC	PeacoQC	1.10.0	PeacoQC()
5	remove_doublets	CytoPipeline	1.0.0	removeDoubletsCytoPipeline()
6	remove_debris	flowClust	3.38.0	tmixFilter()
7	remove_dead_cells	flowDensity	1.34.0	deGate()

Table S2. PeacoQC-based pipeline pre-processing steps.

Step	Name	R package	Version	Method
1	flowframe_read	flowCore	2.12.0	read.FCS()
2	remove_margins	PeacoQC	1.10.0	removeMargins()
3	perform_QC	flowAI	1.30.0	flow_auto_qc()
4	compensate	flowCore	2.12.0	compensate()
5	remove_doublets	CytoPipeline	1.0.0	removeDoubletsCytoPipeline()
6	remove_debris	flowClust	3.38.0	tmixFilter()
7	remove_dead_cells	flowDensity	1.34.0	deGate()

Table S3. flowAI-based pipeline pre-processing steps.

Supplementary Figures

```
library("CytoPipeline") # main package
library("CytoPipelineUtils") # adds wrappers to external packages methods

rawDataDir <- "/rawData" # input raw data directory here
jsonDir <- "/pipeDefs" # input pipeline definition directory here

QCMethod <- "PeacoQC" # choose which pipeline to run
#QCMethod <- "FlowAI"

expName <- paste0("HBVT_ExVivo_", QCMethod)
jsonFile <- file.path(jsonDir,
  paste0("HBVT_ExVivo_", QCMethod, ".json"))

# locates the 55 raw fcs files
sampleFiles <- list.files(path = rawDataDir, pattern = "*.fcs")

# creates CytoPipeline object
pipl <- CytoPipeline(jsonFile,
  experimentName = expName,
  sampleFiles = sampleFiles)

# executes pipeline
execute(pipl)

# monitors the state of the processing queues after the run
plotCytoPipelineProcessingQueue(
  pipl,
  whichQueue = "scale transform")

plotCytoPipelineProcessingQueue(
  pipl,
  whichQueue = "pre-processing",
  sampleFile = 33) # input specific sample file of which to check the run

# interactive visualization using shiny app
library("CytoPipelineGUI")
CytoPipelineGUI::CytoPipelineCheckApp()
```

Figure S1. Simple sample R code to define a *CytoPipeline* object, run it and visualize the results.



Figure S2. Illustration of the colour code used to represent the pipeline workflows. Green steps are the ones that have already been run and produced results, indicating a non problematic run. Orange steps are the ones that are correctly defined, but have not run yet for the displayed sample file. Red steps show an inconsistency problem between the already stored results, and the *CytoPipeline* object, meaning that the pipeline was previously run, either with a different number of steps, or with one or several steps that used a different method or different set of parameters. On the above picture, part A shows a situation where part of the pipeline has fully run. Part B shows a situation of pipeline that has run, but not all steps. Part C shows an inconsistency between the pipeline definition and the results stored.



Figure S3. Screenshot of the interactive GUI application allowing to compare flow frames, and implemented in the *CytoPipelineGUI* package. Here the displayed plots correspond to Figure 4 of the main text.

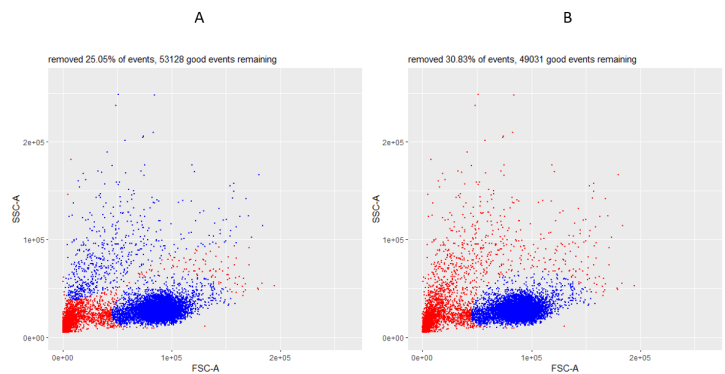


Figure S4. Comparison between the outcome of the debris removal step between the *PeacoQC*-based pipeline with 3 clusters (A), and the *PeacoQC*-based pipeline with 2 clusters (B). The latter setting better eliminates undesirable events than the former.

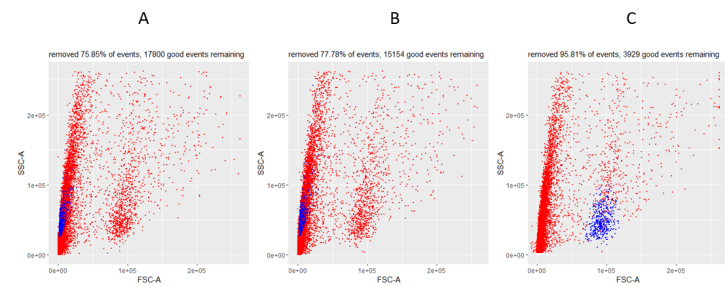


Figure S5. Comparison between the outcome of three combined steps (doublets, debris and dead cells removals) between the *PeacoQC*-based pipeline (A), the *flowAI*-based pipeline (B), and the ground truth (C), on sample *D93_A05*. Due to the bad quality of this sample, i.e. the majority of the events were composed of debris and dead cells, both automatic pipelines were unable to select the right population of lymphocytes.

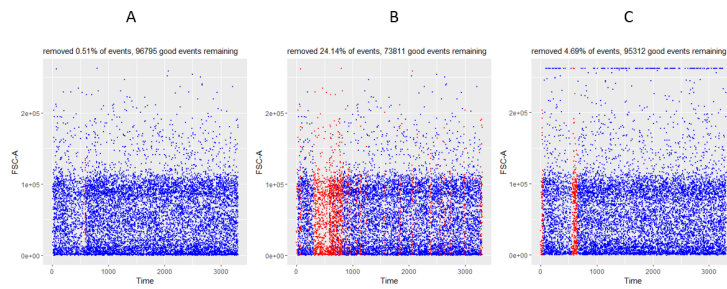


Figure S6. Comparison between the QC in time step with *PeacoQC* (A), with *flowAI* (B), and the time gate applied in manual gating (ground truth) (C), on sample D91_D03. *flowAI* removes more event than the ground truth, while *PeacoQC* removes less events.

Supplementary text file 1: json file describing the *PeacoQC*-based pre-processing pipeline

```
{
  "flowFramesPreProcessingSteps": [
    {
      "name": ["flowframe_read"],
      "RUN": ["readSampleFiles"],
      "ARGS": {
        "truncate_max_range": [false],
        "min.limit": null
      }
    },
    {
      "name": ["remove_margins"],
      "RUN": ["removeMarginsPeacoQC"],
      "ARGS": {
        "channelSpecifications": {
          "AllFluoChannels": [-300, 262144]
        }
      }
    },
    {
      "name": ["compensate"],
      "RUN": ["compensateFromMatrix"],
      "ARGS": {
        "matrixSource": ["pData"],
        "pDataVar": ["day"],
        "pDataPathMapping": {
          "D91": ["./compensation/Compensations Liver D91.csv"],
          "D93": ["./compensation/Compensations Liver D93.csv"]
        }
      }
    },
    {
      "name": ["perform_QC"],
      "RUN": ["qualityControlPeacoQC"],
      "ARGS": {
        "preTransform": [true],
        "min_cells": [150],
        "max_bins": [500],
        "step": [500],
        "nWD": [6],
        "IT_limit": [0.6],
        "force_IT": [150],
        "peak_removal": [0.3333],
        "min_nr_bins_peakdetection": [10]
      }
    },
    {
      "name": ["remove_doublets"],
      "RUN": ["removeDoubletsCytoPipeline"],
      "ARGS": {
        "areaChannels": ["FSC-A"],
        "heightChannels": ["FSC-H"],
        "nmds": [3]
      }
    },
    {
      "name": ["remove_debris"],
      "RUN": ["removeDebrisFlowClustTmix"],
      "ARGS": {
        "FSCChannel": ["FSC-A"],
        "SSCChannel": ["SSC-A"],
        "nClust": [2],
        "level": [0.97],
        "B": [100],
        "verbose": [true]
      }
    },
    {
      "name": ["remove_dead_cells"],
      "RUN": ["removeDeadCellsDeGate"],
      "ARGS": {
        "preTransform": [true],
        "LDMarker": ["Live & Dead"]
      }
    }
  ],
  "scaleTransformProcessingSteps": [
    {
      "name": ["flowframe_read"],
      "RUN": ["readSampleFiles"],
      "ARGS": {
        "whichSamples": ["random"],
        "nSamples": [4],
        "seed": [0],
        "truncate_max_range": [false],
        "min.limit": null
      }
    }
  ]
}
```

```

    },
    {
      "name": ["remove_margins"],
      "RUN": ["removeMarginsPeacoQC"],
      "ARGS": {
        "channelSpecifications": {
          "AllFluoChannels": [-300, 262144]
        }
      }
    },
    {
      "name": ["compensate"],
      "RUN": ["compensateFromMatrix"],
      "ARGS": {
        "matrixSource": ["pData"],
        "pDataVar": ["day"],
        "pDataPathMapping": {
          "D91": ["./compensation/Compensations Liver D91.csv"],
          "D93": ["./compensation/Compensations Liver D93.csv"]
        }
      }
    },
    {
      "name": ["flowframe_aggregate"],
      "RUN": ["aggregateAndSample"],
      "ARGS": {
        "nTotalEvents": [10000],
        "seed": [0]
      }
    },
    {
      "name": ["scale_transform_estimate"],
      "RUN": ["estimateScaleTransforms"],
      "ARGS": {
        "fluoMethod": ["estimateLogicle"],
        "scatterMethod": ["linearQuantile"],
        "scatterRefMarker": ["CD4"]
      }
    }
  ]
}

```


Supplementary text file 2: json file describing the *flowAI*-based pre-processing pipeline

```
{
  "flowFramesPreProcessingSteps": [
    {
      "name": ["flowframe_read"],
      "RUN": ["readSampleFiles"],
      "ARGS": {
        "truncate_max_range": [false],
        "min_limit": null
      }
    },
    {
      "name": ["remove_margins"],
      "RUN": ["removeMarginsPeacoQC"],
      "ARGS": {
        "channelSpecifications": {
          "AllFluoChannels": [-300, 262144]
        }
      }
    },
    {
      "name": ["perform_QC"],
      "RUN": ["qualityControlFlowAI"],
      "ARGS": {
        "preTransform": [false],
        "remove_from": ["all"],
        "second_fractionFR": [0.1],
        "deviationFR": ["MWD"],
        "alphaFR": [0.01],
        "decompFR": [true],
        "outlier_binsFS": [false],
        "pen_valueFS": [500],
        "max_cptFS": [3],
        "sideFM": ["both"],
        "neg_valuesFM": [1]
      }
    },
    {
      "name": ["compensate"],
      "RUN": ["compensateFromMatrix"],
      "ARGS": {
        "matrixSource": ["pData"],
        "pDataVar": ["day"],
        "pDataPathMapping": {
          "D91": ["./compensation/Compensations Liver D91.csv"],
          "D93": ["./compensation/Compensations Liver D93.csv"]
        }
      }
    },
    {
      "name": ["remove_doublets"],
      "RUN": ["removeDoubletsCytoPipeline"],
      "ARGS": {
        "areaChannels": ["FSC-A"],
        "heightChannels": ["FSC-H"],
        "nmads": [3]
      }
    },
    {
      "name": ["remove_debris"],
      "RUN": ["removeDebrisFlowClustTmix"],
      "ARGS": {
        "FSCChannel": ["FSC-A"],
        "SSCChannel": ["SSC-A"],
        "nClust": [2],
        "level": [0.97],
        "B": [100],
        "verbose": [true]
      }
    },
    {
      "name": ["remove_dead_cells"],
      "RUN": ["removeDeadCellsDeGate"],
      "ARGS": {
        "preTransform": [true],
        "LDMarker": ["Live & Dead"]
      }
    }
  ],
  "scaleTransformProcessingSteps": [
    {
      "name": ["flowframe_read"],
      "RUN": ["readSampleFiles"],
      "ARGS": {
        "whichSamples": ["random"],
        "nSamples": [4],
        "seed": [0],
        "truncate_max_range": [false],
        "min_limit": null
      }
    }
  ]
}
```

```

    },
    {
      "name": ["remove_margins"],
      "RUN": ["removeMarginsPeacoQC"],
      "ARGS": {
        "channelSpecifications": {
          "AllFluoChannels": [-300, 262144]
        }
      }
    },
    {
      "name": ["compensate"],
      "RUN": ["compensateFromMatrix"],
      "ARGS": {
        "matrixSource": ["pData"],
        "pDataVar": ["day"],
        "pDataPathMapping": {
          "D91": ["./compensation/Compensations Liver D91.csv"],
          "D93": ["./compensation/Compensations Liver D93.csv"]
        }
      }
    },
    {
      "name": ["flowframe_aggregate"],
      "RUN": ["aggregateAndSample"],
      "ARGS": {
        "nTotalEvents": [10000],
        "seed": [0]
      }
    },
    {
      "name": ["scale_transform_estimate"],
      "RUN": ["estimateScaleTransforms"],
      "ARGS": {
        "fluoMethod": ["estimateLogicle"],
        "scatterMethod": ["linearQuantile"],
        "scatterRefMarker": ["CD4"]
      }
    }
  ]
}

```

Appendix C

Supplementary results for Chapter 8

Supplementary Materials

Supplementary Tables

Donor ID	Group	Former data acquisition	Later data acquisition
1	Younger Adult	F1	L1
2	Younger Adult		L2
3	Younger Adult		L3
4	Younger Adult	F4	L4
5	Younger Adult	F5	L5
6	Younger Adult	F6	
7	Younger Adult	F7	
11	Older Adult		L11
12	Older Adult		L12
13	Older Adult		L13
14	Older Adult	F14	L14
15	Older Adult		L15
16	Older Adult	F16	
17	Older Adult	F17	
18	Older Adult	F18	

Table S1. Sample groups and data acquisitions for the *ImmunoSenescence Human PBMC* dataset.

Supplementary Figures

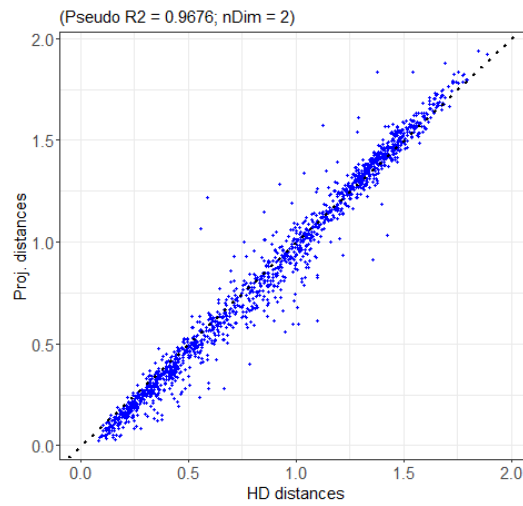


Figure S1. Shepard's diagram for the projection of the *HBV Chronic Mouse* dataset. *CytoMDS* required 2 dimensions to reach the $\text{pseudo } R^2$ quality threshold of 0.95. Each dot represents a sample pair, where the x value is the high dimensional *Earth Mover's Distance* between the two samples and the y value is the obtained euclidian distance on the low dimensional projection. A projection of high quality results in most points being close to the ideal 45 degree identity line.

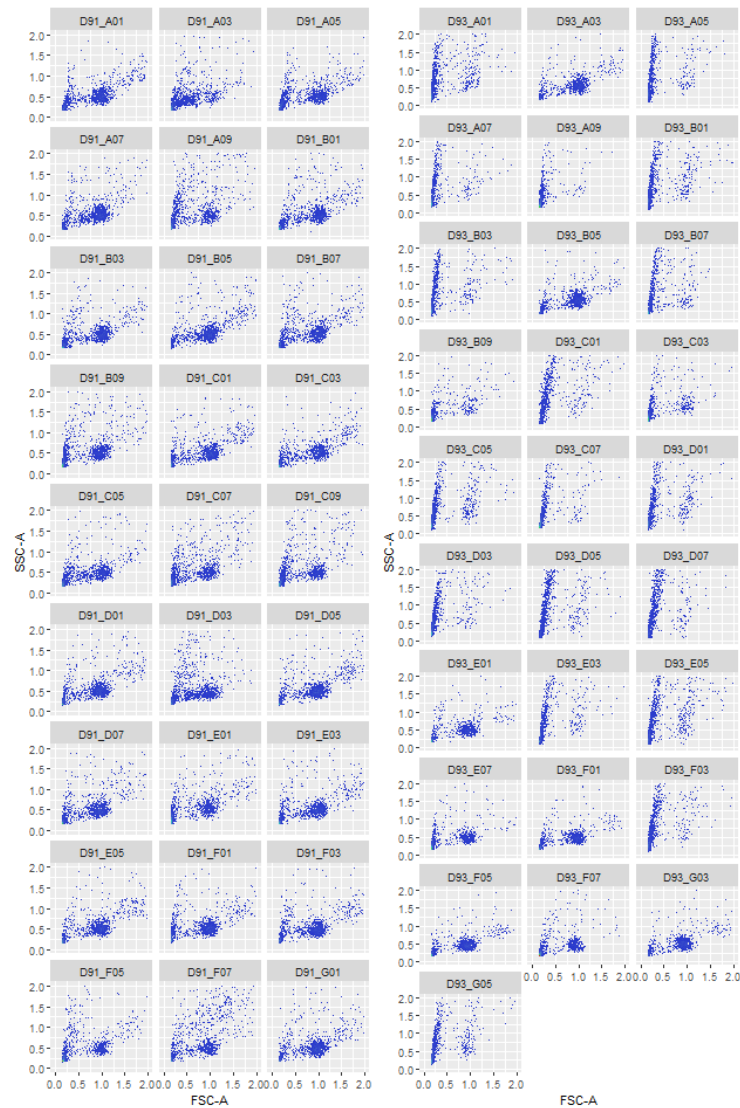


Figure S2. FSC-A vs. SSC-A 2D visualization of the 55 samples of the HBV Chronic Mouse dataset, revealing a clear difference of pattern between low quality samples and good quality samples, corresponding to the two sample groups in Figure 2. Compared to Figure 2, here the axes have undergone a linear mapping to the [0.0, 2.0] range, in order to keep the axis legends readable.

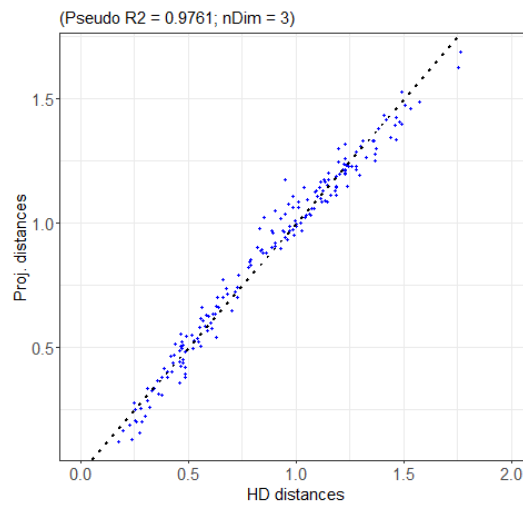


Figure S3. Shepard's diagram for the projection of the *ImmunoSenescence Human PBMC* dataset. CytoMDS required 3 dimensions to reach the *pseudo R*² quality threshold of 0.95. Each dot represents a sample pair, where the x value is the high dimensional *Earth Mover's Distance* between the two samples and the y value is the obtained euclidian distance on the low dimensional projection. A projection of high quality results in most points being close to the ideal 45 degree identity line.

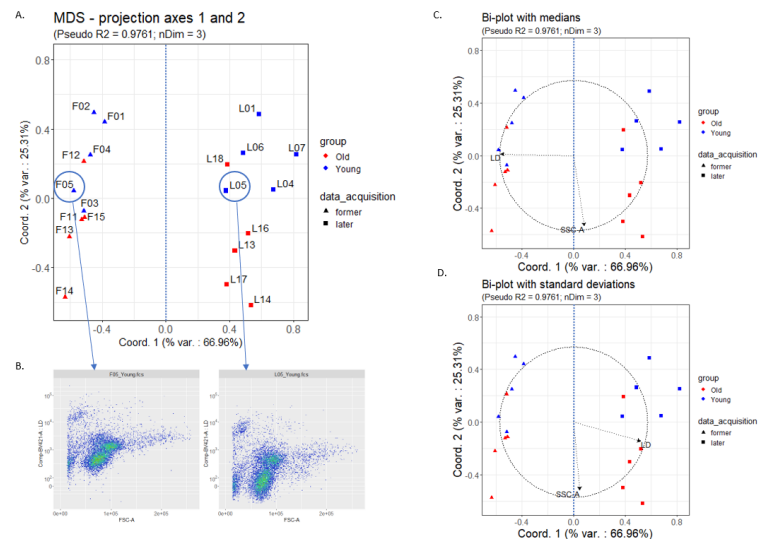


Figure S4. Low dimensional projection of the *ImmunoSenescence Human PBMC* dataset, 2D projection on axes 1 and 2. The projection plot (A) reveals a clear separation, along the first axis, between the samples of the former data acquisition (triangles), and the samples of the later data acquisition (squares). This is an obvious batch effect. According to the bi-plots using channel medians (C) and channel standard deviations (D), we can attribute this batch effect mainly to location and scale of *Live/Dead* channel. This is confirmed when plotting 2D representations of selected samples having opposite first coordinates (B).

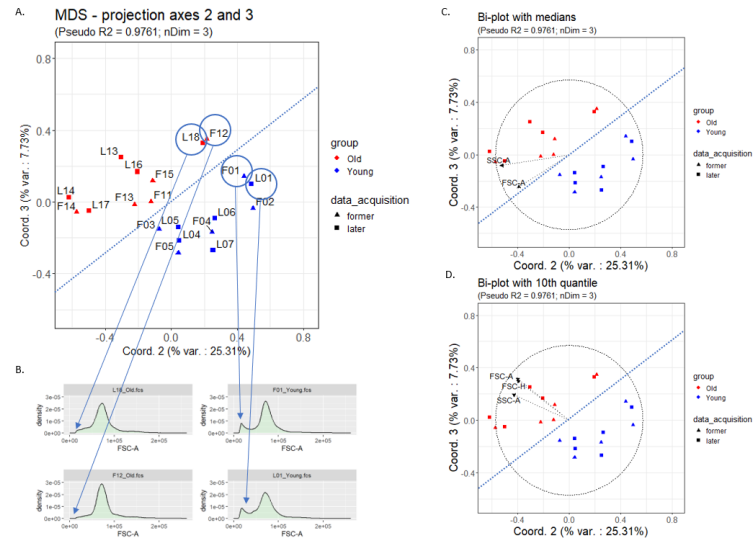


Figure S5. Low dimensional projection of the *ImmunoSenescence Human PBMC* dataset, 2D projection on axes 2 and 3. On the projection plot (A), the two groups 'old' in red vs. 'young' in blue, can be separated along a direction that is a linear combination of axes 2 and 3. Bi-plots using channel medians (C) does not highlight any channel median strongly correlated with the direction of separation. However, a bi-plot with a well-chosen statistics, i.e. the 10th percentile, enables the identification of a strong correlation with this quantile for the FSC and SSC channels. This can be confirmed when comparing the marginal distributions of the FSC-A channel for selected samples from the two sides of the border (B).

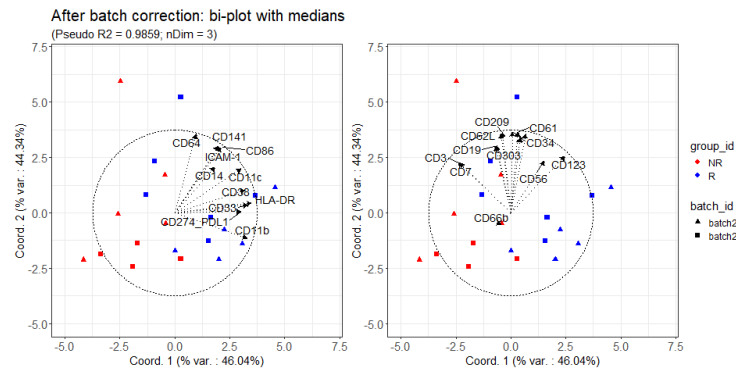


Figure S6. Bi-plots of the projection of the *Krieg_Anti_PD1* dataset on the first two axes, after batch correction, and representing all channel medians, split into two groups. In the original study (Krieg et al., 2018), it was highlighted that the frequency of a small subpopulation of CD14⁺ CD33⁺ HLA-DR^{hi} ICAM-1⁺ CD64⁺ CD141⁺ CD86⁺ CD11c⁺ CD38⁺ PD-L1⁺ CD11b⁺ monocytes in baseline samples was positively associated with responder status following immunotherapy treatment. On the left plot, we selected these 11 markers and on the right plot, the other 14 markers are represented for comparison. The intensity of the markers of the first group tends to be more correlated with the direction separating responders (blue) and non responders (red).

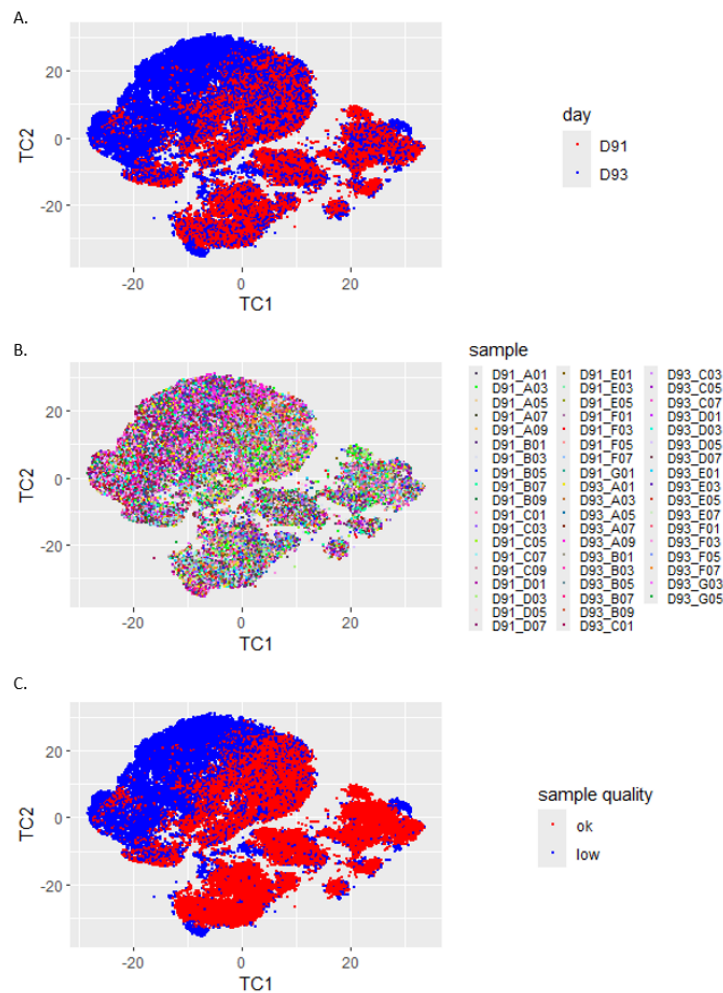


Figure S7. tSNE of the *HBV Chronic Mouse* dataset. Input data consist of a sub-sampling of 50,000 random events from the 55 samples, with all 12 markers included in the calculation. Prior to sub-sampling, channel intensities were transformed using respectively bi-exponential transformations for fluorescent channels, and linear transformations for *FSC/SSC*, as described in the Results section. Highlighting the displayed events by acquisition day (A) suggests the existence of a corresponding batch effect. However a substantial amount of the blue dots are mixed with the red ones, because the effect is linked to a quality issue affecting only a subset of D93 samples. Highlighting the displayed events by sample (B) is uninformative, due to the high number of samples, and high number of events per sample. (C) Colouring the events by quality cluster - which was identified thanks to *CytoMDS* analysis - seems to show less residual dots mixing than in (A).

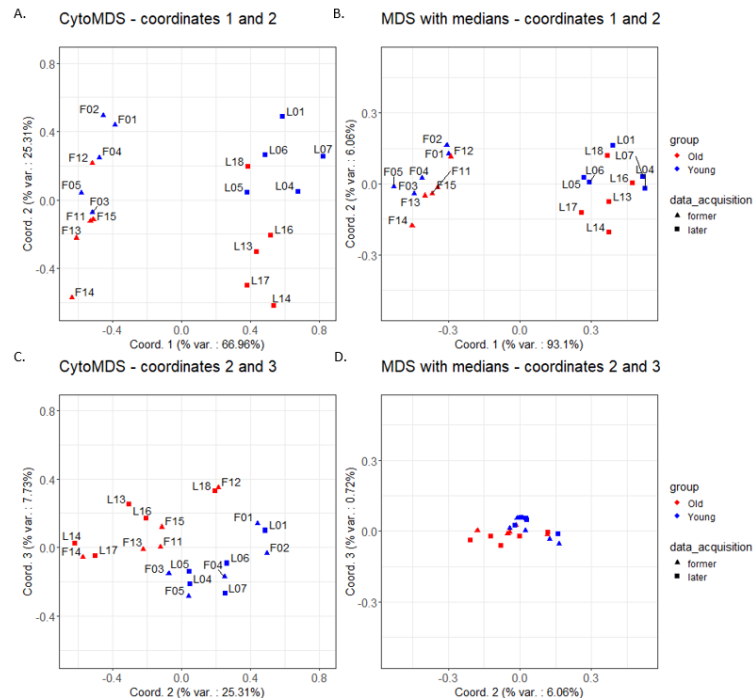


Figure S8. Low dimensional projections of the *ImmunoSenescence Human PBMC* dataset. The above plots show a comparison of the projection on axes 1 and 2 of respectively the *CytoMDS* method (A) - which uses *Earth Mover's Distances* to assess dissimilarity between samples - and *MDS* using channel medians as input (B). The two plots show similar patterns. Similarly, plots (C) and (D) show the same comparison, now using axes 2 and 3. Here, *CytoMDS* projection (C) shows a visible separation between the two sample groups. However, on the right plot (D), there is little variability left between data points. This difference can be explained by the fact that the direction of separation between the two sample groups in the *CytoMDS* projection could be associated to the 10th quantile of the *FSC* and *SSC* channel distributions (Figure S5), but not to channel medians.

Supplementary Results - benchmarking of CPU time for *EMD* computation

In this section, we present a benchmarking of the CPU time required to compute the *EMD* between two cytometry samples, as a function of number of markers and events. To perform this study, we randomly selected two cytometry samples from the *ImmunoSenescence Human PBMC* dataset, and we pre-processed these two samples by applying compensation and scale transformation, as described in the Results section. From these two pre-processed samples, we created two large simulated samples by sampling 10,000,000 events with replacement from the original sample event sets, while keeping the original 28 markers - 26 fluorescent markers and (FSC-A, SSC-A). Finally we added a gaussian random noise with a low standard deviation (0.05) to the scaled intensities.

With these two large simulated reference samples, we let *CytoMDS* calculate their *EMD*, which we considered their reference *EMD*. Then, we launched computational experiments by re-calculating the *EMD* for various combinations of randomly selected subsets of markers or events, and recorded the CPU time needed for each *EMD* run. All calculations were performed with a single core on a standard laptop computer with a 11th Gen Intel-Core i7 @ 2.80 Ghz CPU and 8GB of RAM. These computational experiments were repeated 20 times each. The results are summarized on Figures S9 and S10.

Figure S9 (A) shows that the calculation for the full reference samples with 10 million events - last box plot on the right - took slightly less than one minute to run. This reference time estimates an upper bound for a realistic *EMD* calculation time, since 10 million events and 28 markers is a reasonably large sample size in comparison with usual flow cytometry datasets. The plot also shows that the calculation time decreases steadily as a function of decreasing number of events, with a calculation time of less than one second for 100,000 events. For small samples with less than 1,000 events, a lower plateau is observed for the computation time. In Figure S9 (B), we investigate to what extent the *EMD* using a sub-sampling of a decreasing number of events, remains a good approximation of the true reference *EMD*. This plot shows that sub-sampling 100,000 events is more than enough to obtain an excellent approximation. This means that, even for large sample cases, when the user is not willing to wait up to one minute to obtain the *EMD*, they can still use sub-sampling to obtain satisfactory approximations with much shorter calculation times.

Finally, as is shown in Figure S10, the *EMD* calculation time evolves roughly as a linear function of number of markers. This is expected, since *CytoMDS* approximates the *EMD* by the sum of the per marker *EMDs*, each of them being calculated independently (see Methods section).

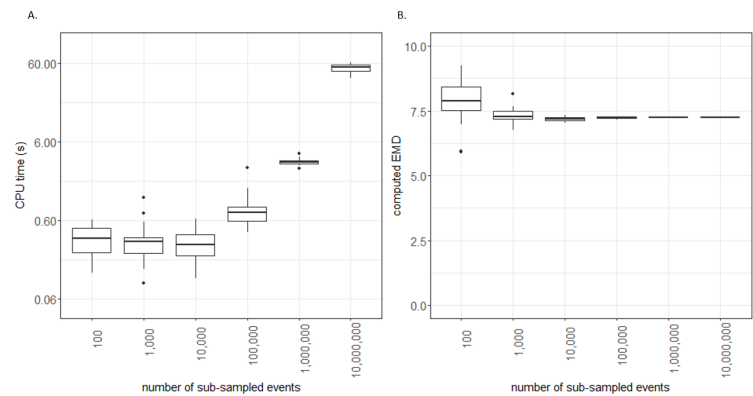


Figure S9. Distribution of calculation time (A) and obtained distance (B) when computing the *EMD* between the two reference samples, as a function of number of events. Each random sub-sampling of events and *EMD* calculation were run 20 times

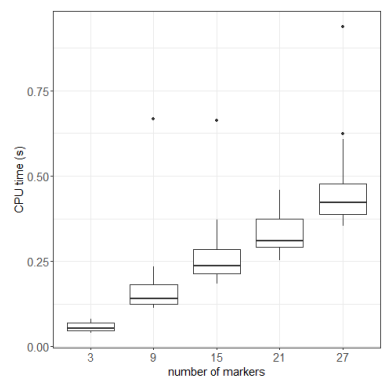


Figure S10. Distribution of calculation time when computing the *EMD* between the two reference samples, as a function of number of included marker, for a sub-sampling of 100,000 events. Each random marker selection and *EMD* calculation were run 20 times

Appendix D

Supplementary results for Chapter 10

D.1 Predicting CMV status with *CytoCNN*: supplementary plots

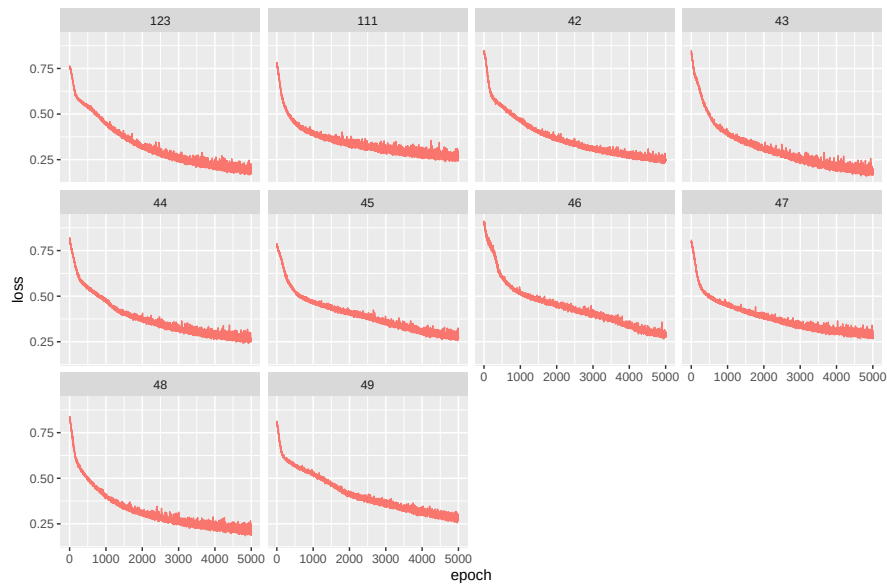


Figure D.1: Evolution of model training loss, as a function of epoch number, for 10 different random seeds.

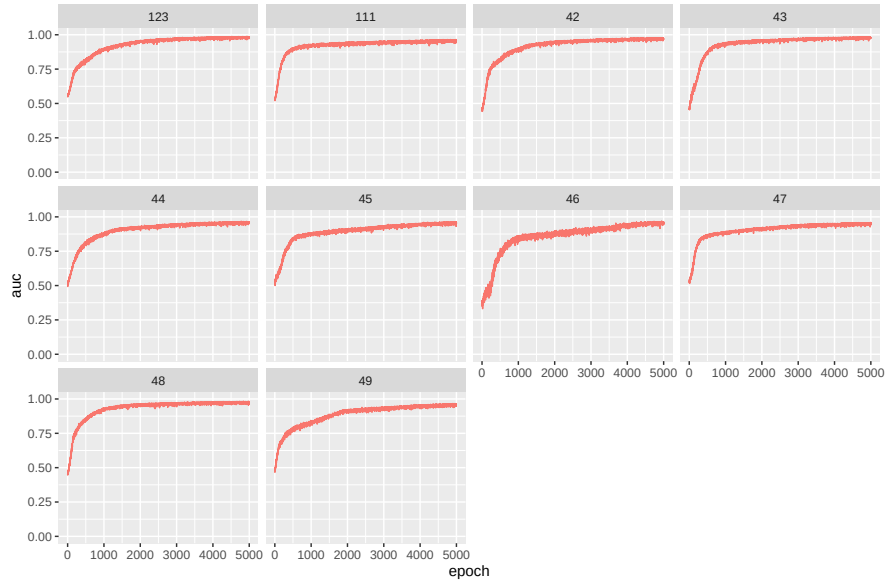


Figure D.2: Evolution of the [AUC](#) in training, as a function of epoch number, for 10 different random seeds.

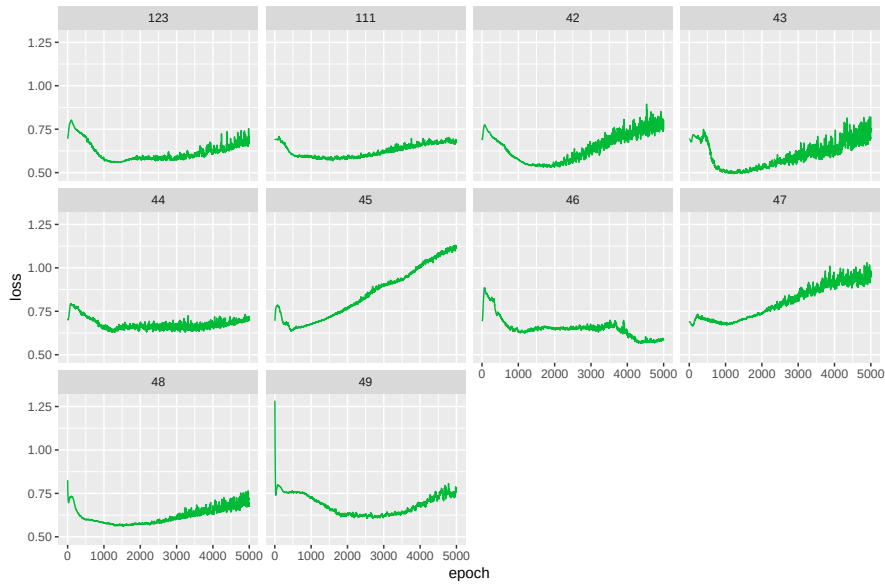


Figure D.3: Evolution of model validation loss, as a function of epoch number, for 10 different random seeds.

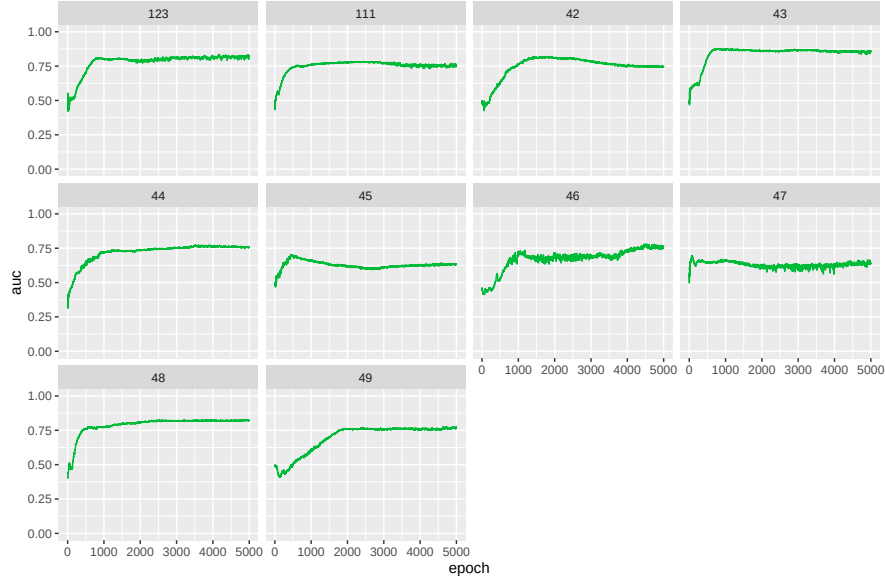


Figure D.4: Evolution of the **AUC** in validation, as a function of epoch number, for 10 different random seeds.

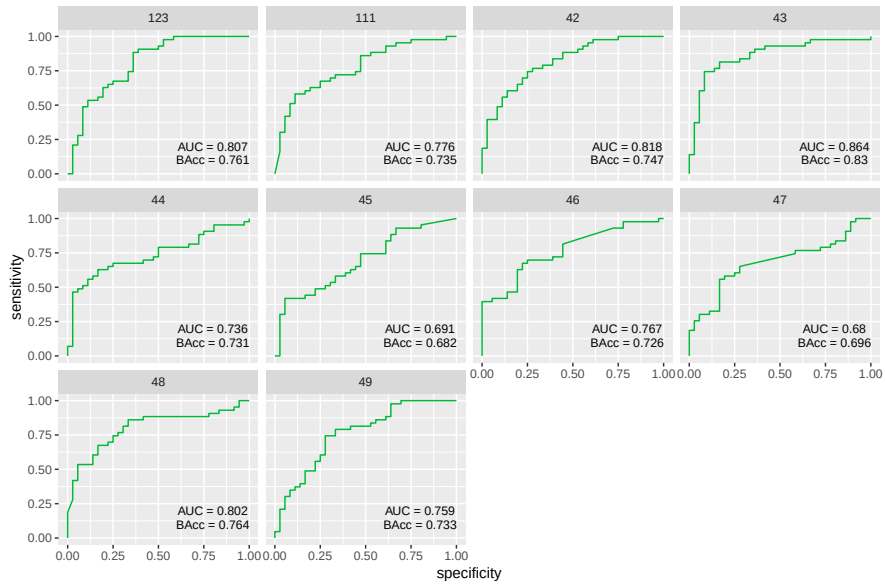


Figure D.5: **ROC** curves obtained in validation, for 10 different random seeds. Curves are annotated with the corresponding **AUC** and **BAcc** values.

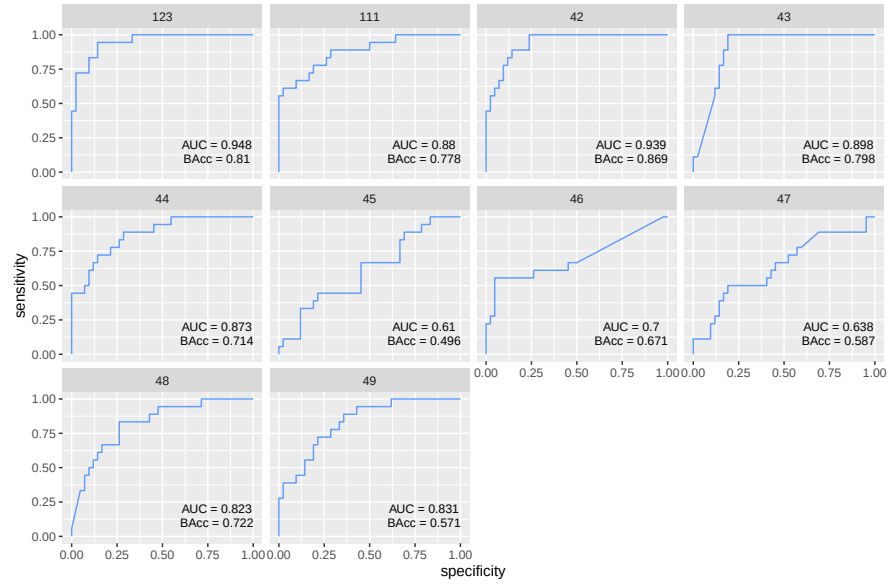


Figure D.6: ROC curves obtained in test, for 10 different random seeds. Curves are annotated with the corresponding AUC and BAcc values.

D.2 Using *CytoCNN* for the *FlowCapIV* dataset - supplementary plots

D.2.1 Base Model

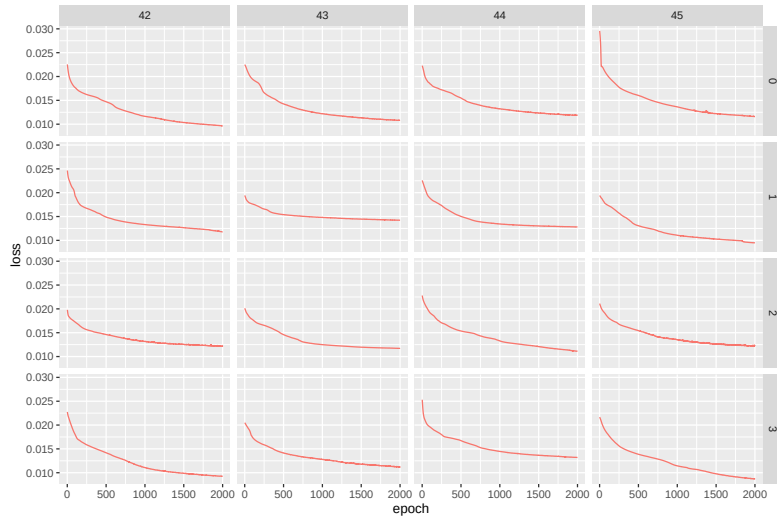


Figure D.7: Base model: evolution of model training loss, as a function of epoch number, for 4 different random seeds, and 4 folds.

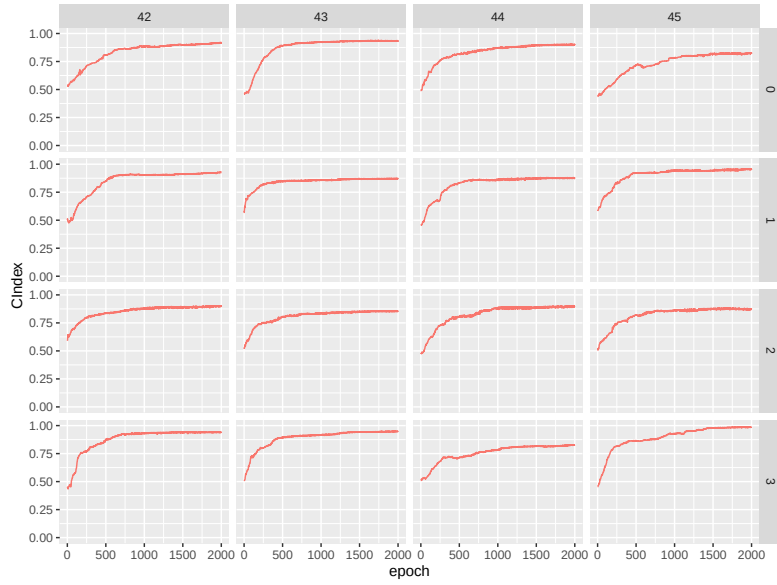


Figure D.8: Base model: evolution of the [C-index](#) in training, as a function of epoch number, for 4 different random seeds, and 4 folds.

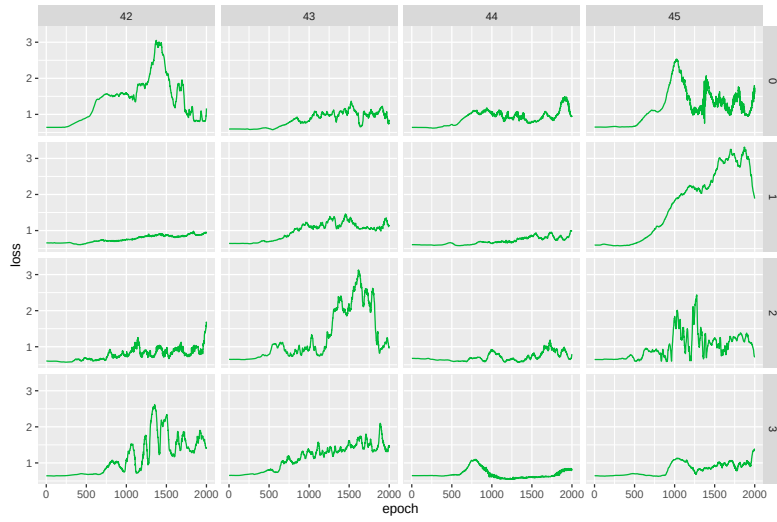


Figure D.9: Base model: evolution of model validation loss, as a function of epoch number, for 4 different random seeds, and 4 folds.



Figure D.10: Base model: evolution of the [C-index](#) in validation, as a function of epoch number, for 4 different random seeds, and 4 folds.

D.2.2 Applying regularisation

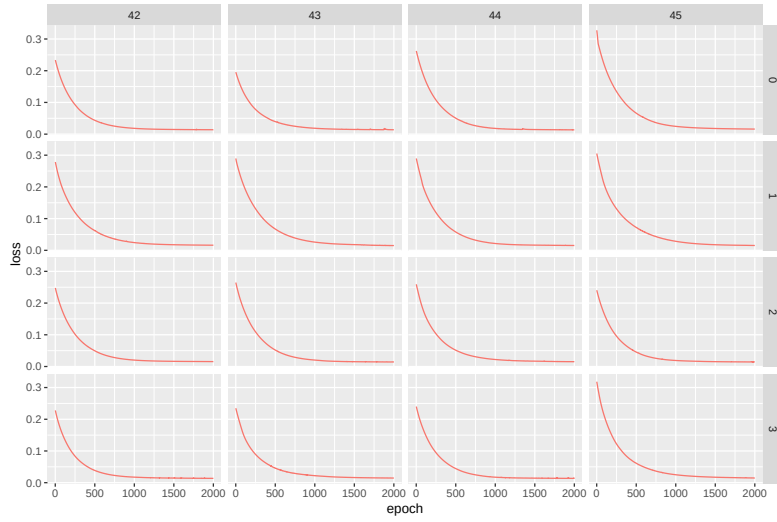


Figure D.11: Applying regularisation: evolution of model training loss, as a function of epoch number, for 4 different random seeds, and 4 folds.

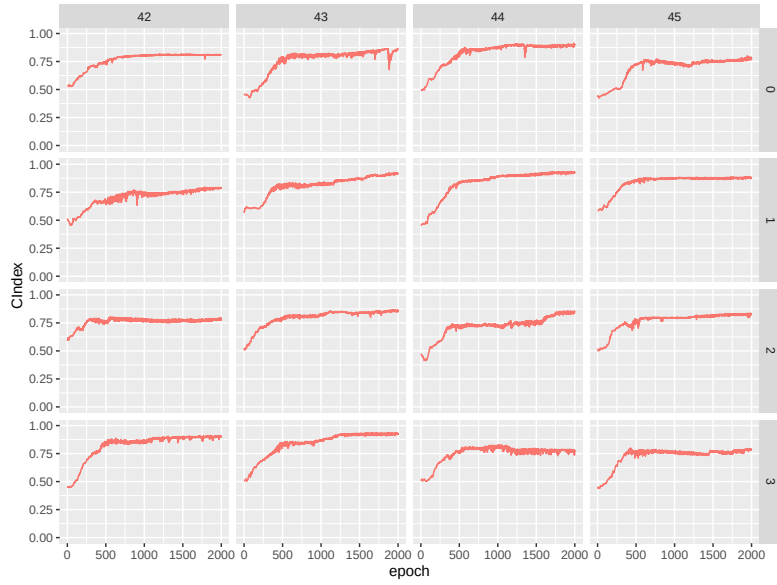


Figure D.12: Applying regularisation: evolution of the [C-index](#) in training, as a function of epoch number, for 4 different random seeds, and 4 folds.

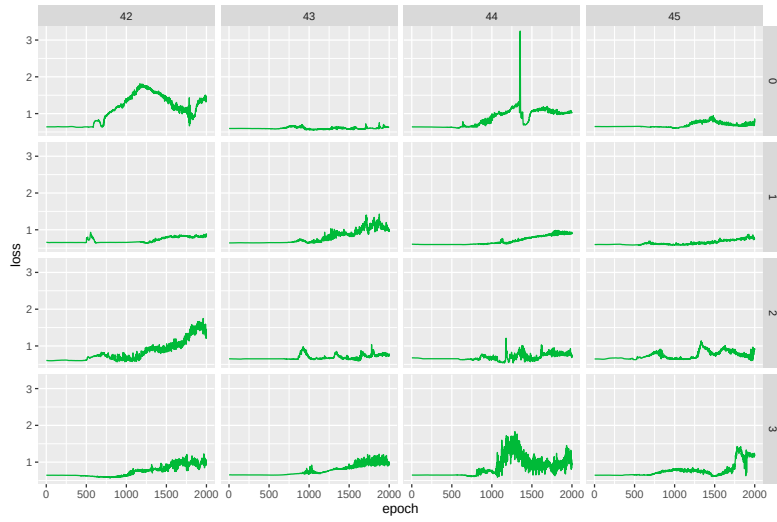


Figure D.13: Applying regularisation: evolution of model validation loss, as a function of epoch number, for 4 different random seeds, and 4 folds.



Figure D.14: Applying regularisation: evolution of the [C-index](#) in validation, as a function of epoch number, for 4 different random seeds, and 4 folds.

D.2.3 Applying dropouts

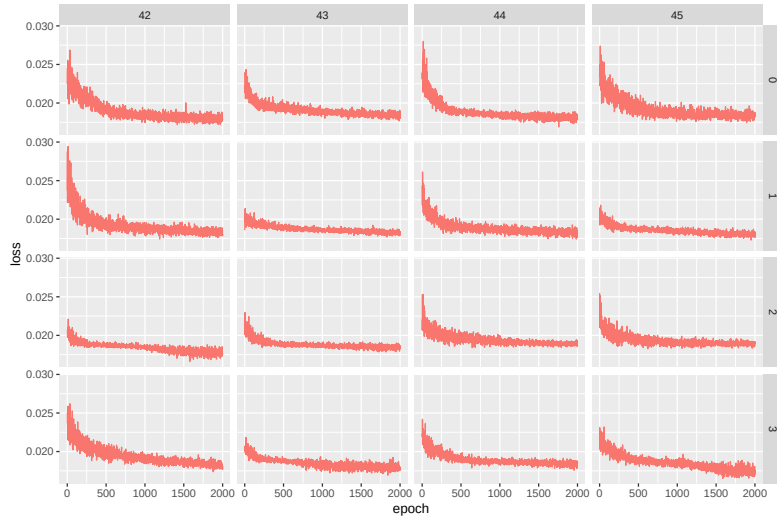


Figure D.15: Applying dropouts: evolution of model training loss, as a function of epoch number, for 4 different random seeds, and 4 folds.

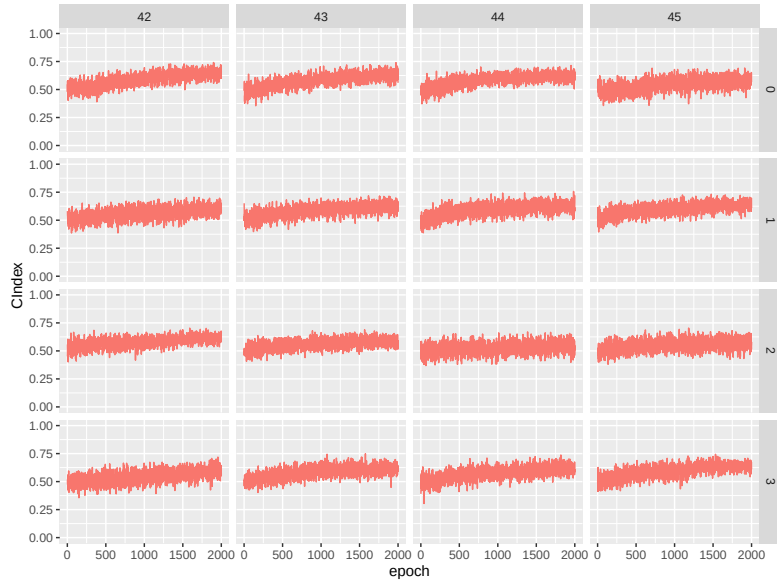


Figure D.16: Applying dropouts: evolution of the [C-index](#) in training, as a function of epoch number, for 4 different random seeds, and 4 folds.

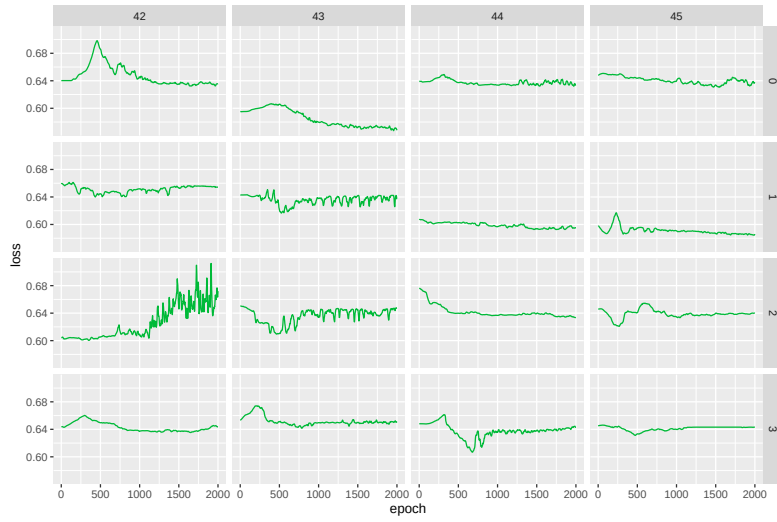


Figure D.17: Applying dropouts: evolution of model validation loss, as a function of epoch number, for 4 different random seeds, and 4 folds.

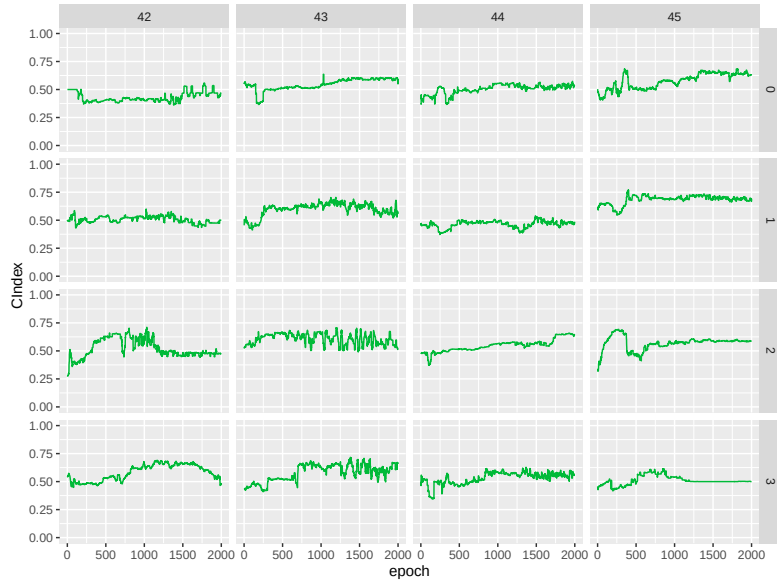


Figure D.18: Applying dropouts: evolution of the **C-index** in validation, as a function of epoch number, for 4 different random seeds, and 4 folds.

D.2.4 Applying data augmentation

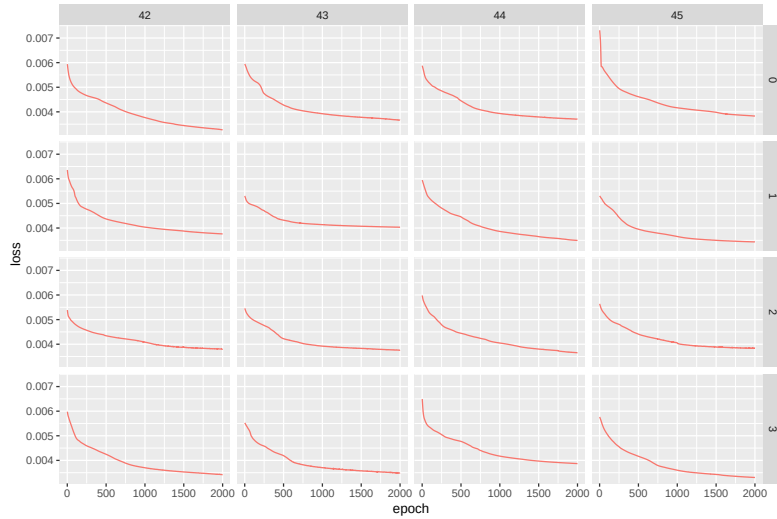


Figure D.19: Applying data augmentation: evolution of model training loss, as a function of epoch number, for 4 different random seeds, and 4 folds.

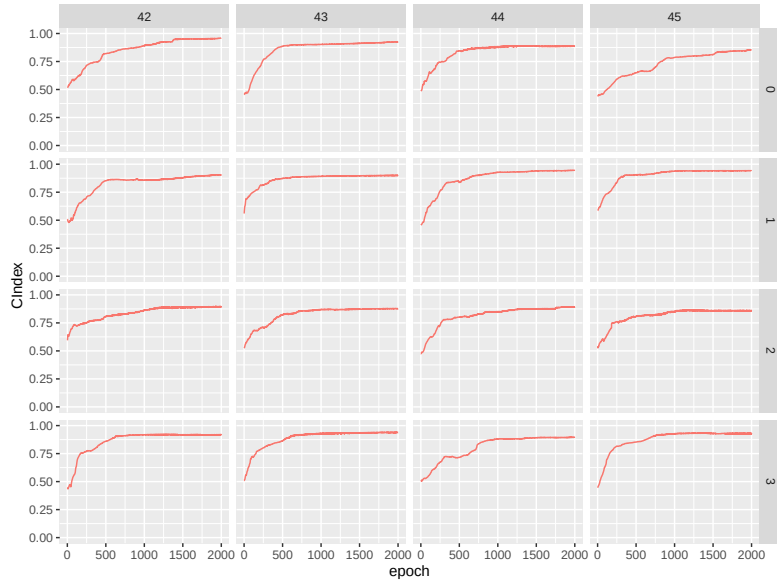


Figure D.20: Applying data augmentation: evolution of the [C-index](#) in training, as a function of epoch number, for 4 different random seeds, and 4 folds.

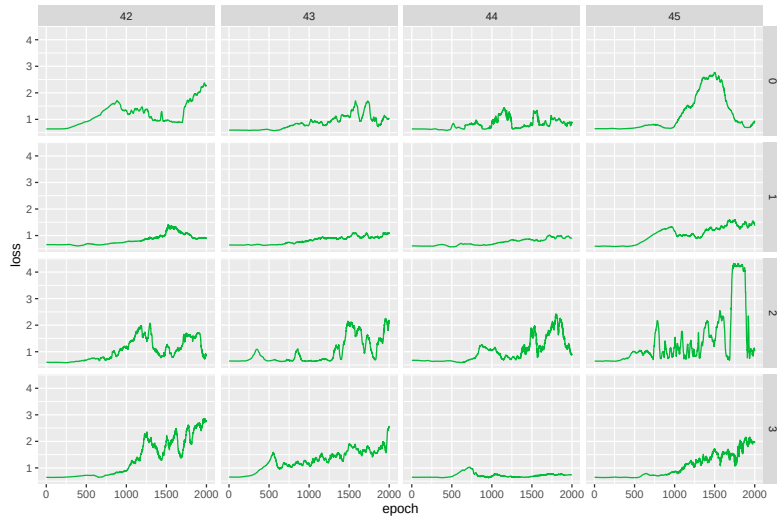


Figure D.21: Applying data augmentation: evolution of model validation loss, as a function of epoch number, for 4 different random seeds, and 4 folds.



Figure D.22: Applying data augmentation: evolution of the [C-index](#) in validation, as a function of epoch number, for 4 different random seeds, and 4 folds.

D.3 Using *CytoCNN* for the *GSK Immunosenescence* dataset - supplementary table and plots

D.3.1 Exploratory QC plots

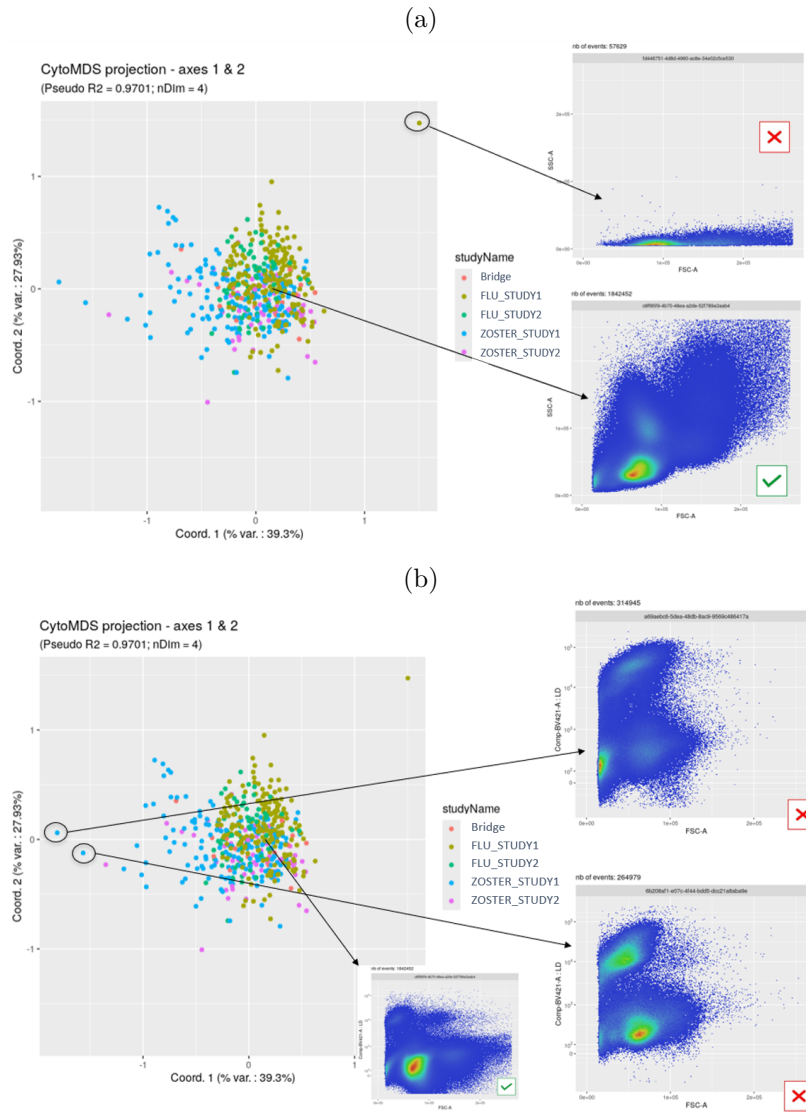


Figure D.23: Identification of 3 outlying samples, using *CytoMDS* projection, followed by visual inspection of 2D plots for these suspicious samples. (a) The top right outlier is compared to a ‘good’ control sample (located in the middle of the plot). There is an issue with SSC-A channel intensities. (b) The two most left-hand samples are compared to the same ‘good’ control sample. These two samples have too much debris and/or dead cells.

D.3.2 Learning curve plots

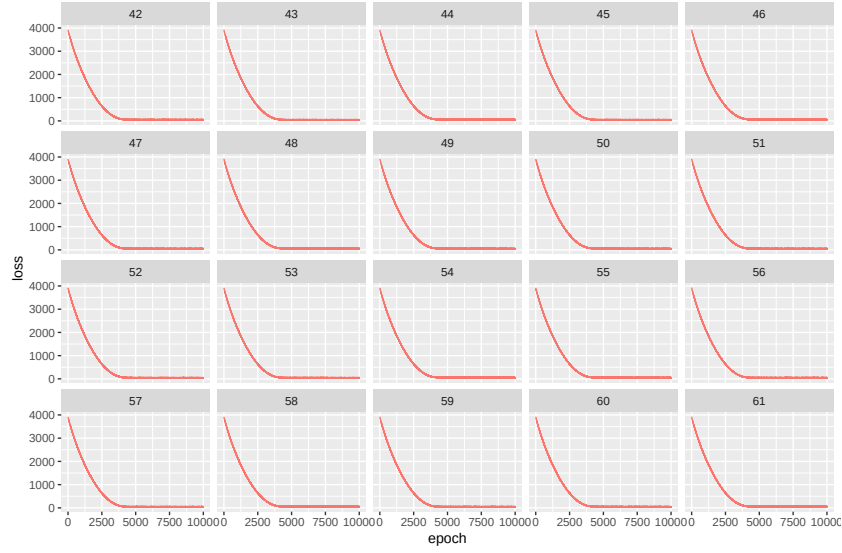


Figure D.24: For the model using L2 regularisation: evolution of model training loss, as a function of epoch number, for 20 different random seeds.

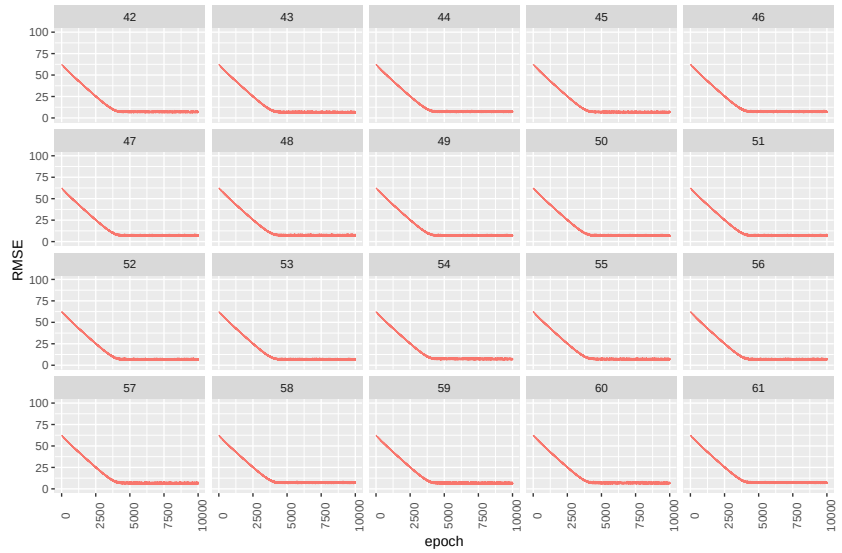


Figure D.25: For the model using L2 regularisation: evolution of model training [RMSE](#), as a function of epoch number, for 20 different random seeds.

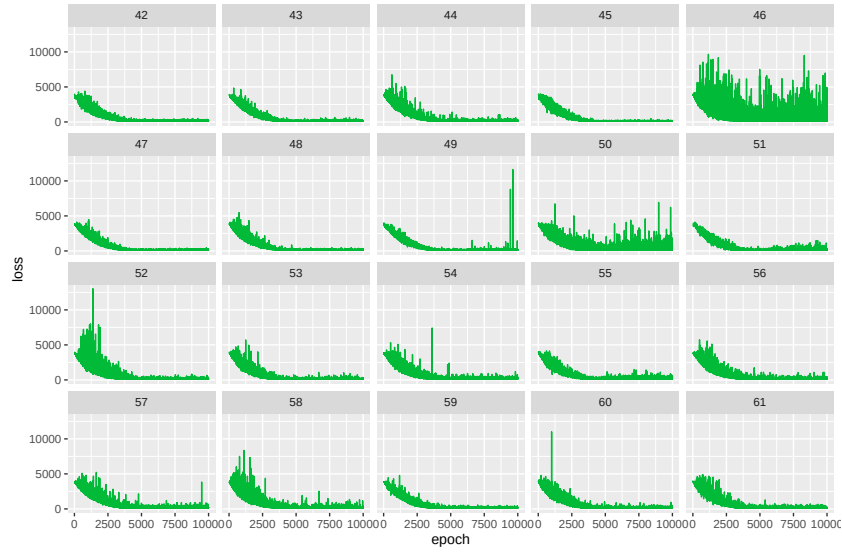


Figure D.26: For the model using L2 regularisation: evolution of model validation loss, as a function of epoch number, for 20 different random seeds.



Figure D.27: For the model using L2 regularisation: evolution of model validation [RMSE](#), as a function of epoch number, for 20 different random seeds.

D.3.3 Model prediction plots

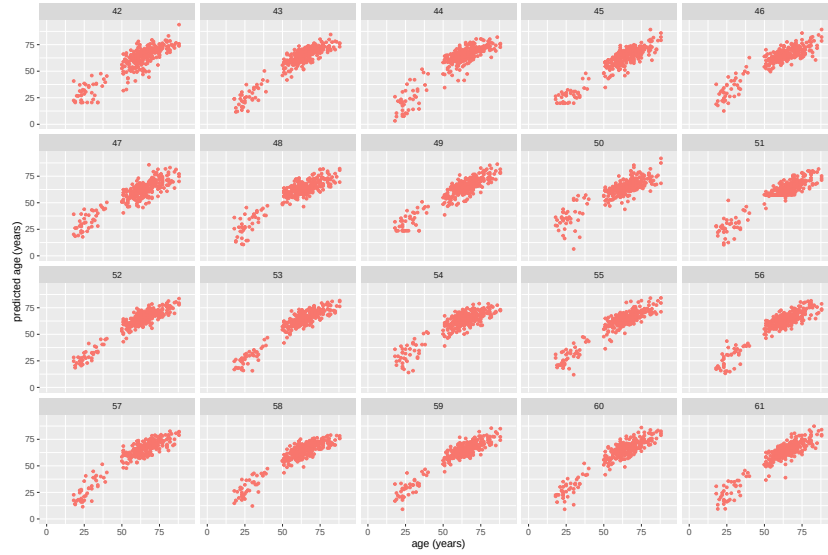


Figure D.28: Scatter plots of model predicted age against true age, for the training set, and 20 different random seeds.

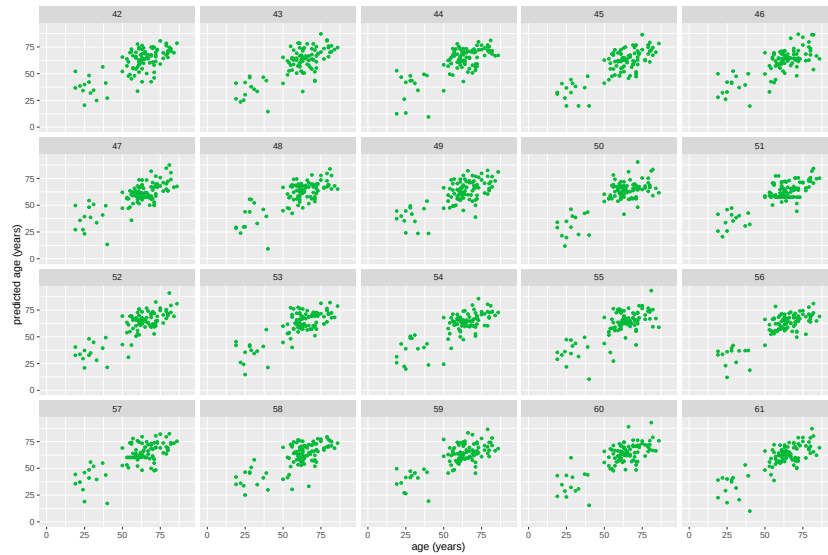


Figure D.29: Scatter plots of model predicted age against true age, for the validation set, and 20 different random seeds.

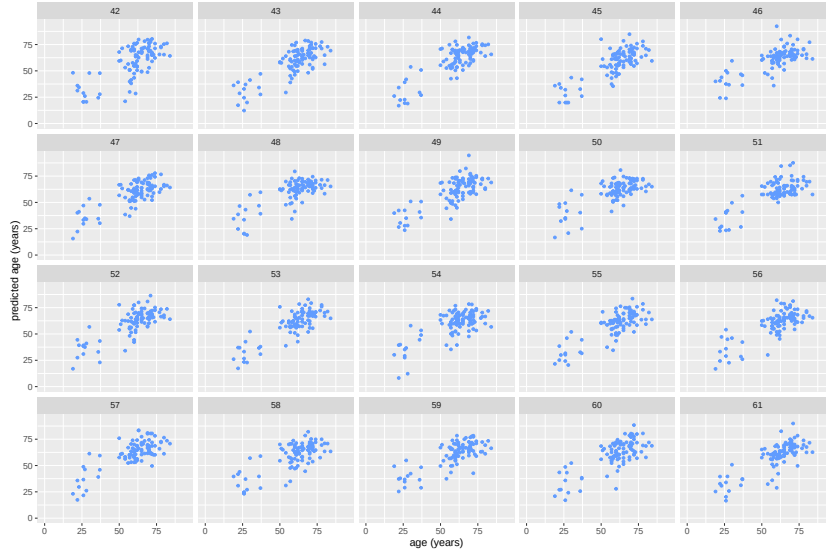


Figure D.30: Scatter plots of model predicted age against true age, for the test set, and 20 different random seeds.

D.3.4 Ensemble model - tables and plots

Metric	Step	Mean	Std-dev	Min	Max
RMSE (years)	train	6.35	0.73	5.38	7.99
	valid	10.17	0.65	8.97	10.94
	test	9.74	0.67	8.86	10.66
MAE (years)	train	5.04	0.54	4.29	6.18
	valid	7.96	0.52	6.87	8.75
	test	7.59	0.59	6.34	9.23
CCC	train	0.91	0.03	0.87	0.94
	valid	0.75	0.03	0.70	0.81
	test	0.76	0.04	0.70	0.83

Table D.1: Statistics for some evaluation metrics in training, validation and test, for 20 models.

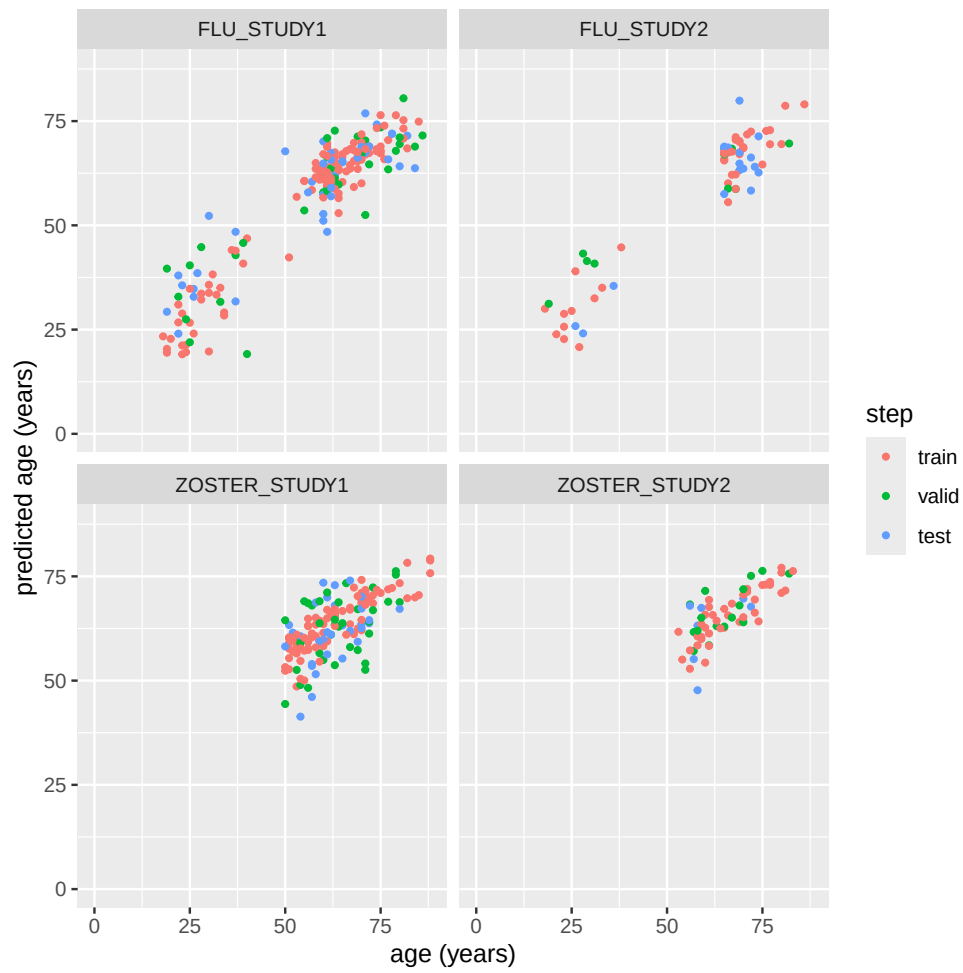


Figure D.31: Scatter plots of ensemble model predicted age against true age, by study, coloured by dataset.



Figure D.32: Scatter plots of ensemble model predicted age against true age, coloured by data acquisition run.

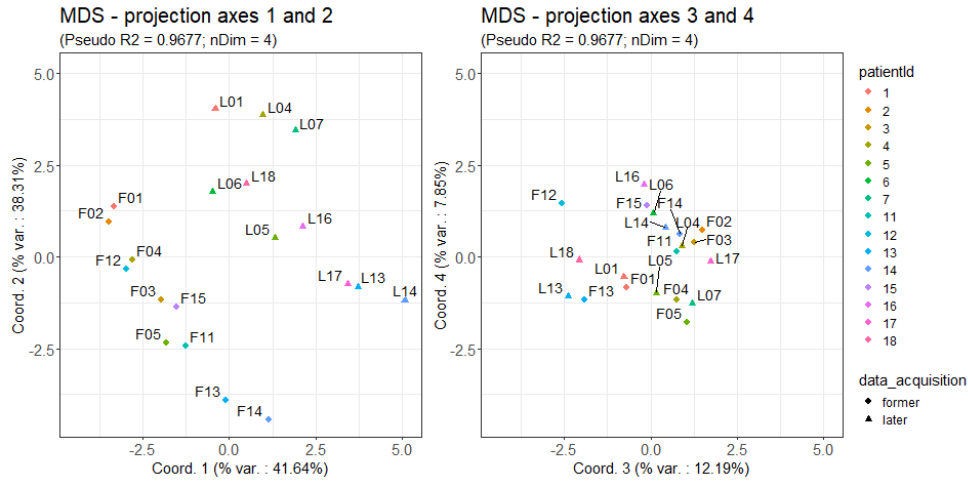


Figure D.33: CytoMDS projection of the panel testing samples using all channels in four dimensions. The left plot (axes 1 and 2) reveals a pronounced batch effect between the earlier and later data acquisitions. When focusing on technical replicate pairs (F1-L1, F4-L4, F5-L5, F13-L13, F14-L14), it is apparent that - apart from the shift corresponding to the batch effect - these pairs occupy nearby coordinates along all other orthogonal dimensions, that is, in the direction perpendicular to the batch effect in the left plot and overall in the right plot.