LOUVAIN
School of Management
RESEARCH INSTITUTE

WORKING PAPER

2015/22

Clustering using a Sum-Over-Forests weighted kernel k-means approach

Felix Sommer,
François Fouss,
Marco Saerens
Louvain School of Management

LOUVAIN SCHOOL OF MANAGEMENT RESEARCH INSTITUTE

Louvain School of Management
Working Paper Series
Editor : Prof. Frank Janssen
(president-ilsm@uclouvain.be)

Clustering using a Sum-Over-Forests weighted kernel k-means approach

Felix Sommer, Louvain School of Management
François Fouss, Louvain School of Management
Marco Saerens, Louvain School of Management

Summary

We develop a weighted kernel k-means approach for clustering using the Sum-Over-Forests density index as node weights. Furthermore, we implement and test the developed algorithm and vary the used kernels to ascertain the algorithms functionality. The algorithm gives good results, but is strongly dependent on the kernel parameters as well as the density index parameter, both of which have to be tuned for optimal results. We achieve a reduction in algorithm runtime, albeit at the price of statistical measure scores.

Keywords : data mining, machine learning, graph theory, clustering, kernel k-means

JEL Classification: C02, C65

Support for this work from the FNRS is gratefully acknowledged.

Corresponding author :
Felix Sommer
Center for Excellence CEMIS
Louvain School of Management / Campus Mons
Université catholique de Louvain
Chaussée de Binche 151
B-7000 Mons, BELGIUM
Email : felix.sommer@uclouvain.be

LSM Working Paper Series

Clustering using a Sum-Over-Forests weighted kernel k-means approach

Felix Sommer
François Fouss
Marco Saerens
Louvain School of Management

December 2015

Contents

1 Introduction

1.1 Motivation

This work discusses a specific graph based k-means clustering approach using the Sum-over-Forests density index as weights. Clustering a graph's nodes into specific partitions is a frequent application in many fields, such as multivariate statistics, data analysis, pattern recognition, data mining or machine learning. The goal is to group a set of objects into subsets, the clusters, such that objects within the cluster share similar properties and are more *alike* than objects not belonging to the cluster.

In clustering, the k-means algorithm has enjoyed enormous popularity both for its straightforward idea and ease of implementation, but more importantly, because at the same time it generates good results. As such, we base our approach on k-means. In particular, we employ a graph-based kernel k-means approach, which is extended by node weights. These weights are derived from the Sum-over-Forests density index (Senelle et al., 2013).

1.2 Algorithm Idea

The idea behind our algorithm is to expand the k-means approach. Based on a kernel k-means with various kernels, we integrate a density index into the calculation to reduce runtime and/or improve the clustering results.

The particular approach is best explained using the flowchart in Figure (1).
We begin with a dataset represented in graph form by means of an adjacency matrix. From this adjacency matrix we calculate two things:

- an arbitrary kernel, such as the commute time or the free energy kernel,
- the graph's density index.

In order to calculate the density index we first calculate the graph's (or the nodes') betweenness and then transform this betweenness into a density using a density evaluation algorithm.

Equipped with the kernel as well as the density index we feed these two objects into our algorithm alongside the original adjacency matrix to perform a weighted kernel

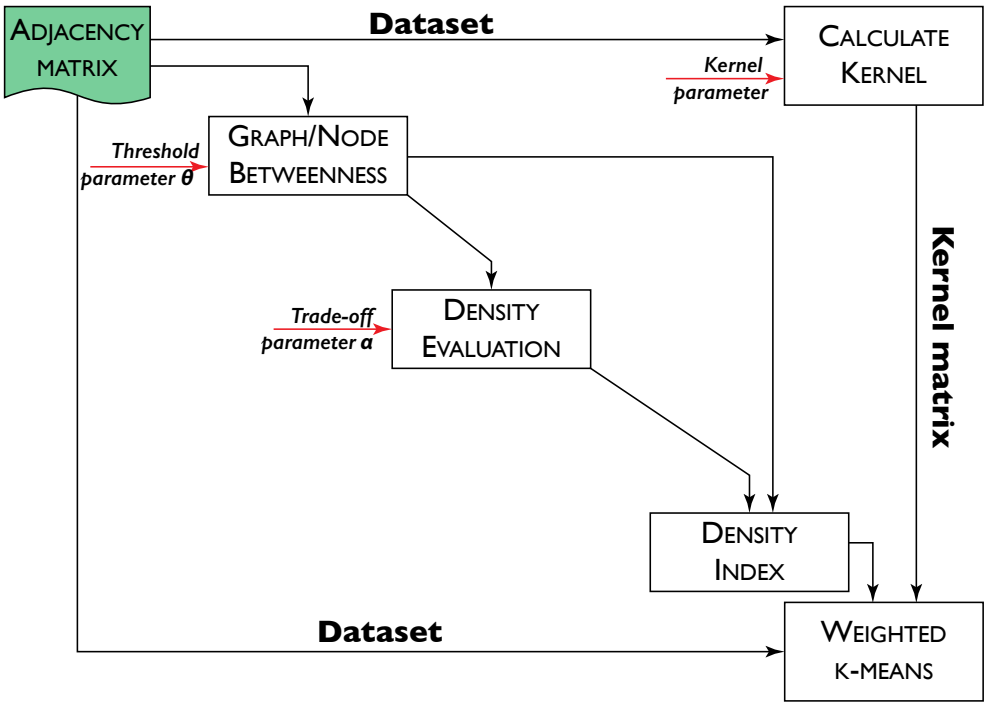


Figure 1: Algorithm flowchart for weighted k-means

k-means clustering. Here, the density index values are used as weights, while the kernel is used as a distance measure of sorts.

Given the results on finding dense regions in graphs in Senelle et al. (2014), we would like to use these features to accelerate or improve clustering. The idea is that nodes in dense regions should (i) attract centroids more quickly and (ii) become "heavy" and as such "attract" other nodes more quickly than an ordinary k-means approach. As such, a k-means solution should converge in fewer iterations as the centroids are bound to move to (and stay at) their final destinations more quickly, thus reducing the algorithm's runtime. Furthermore, less dense nodes should see a greater influence from denser nodes, especially centroids, and thus be assigned to their final centroid as well, again resulting in fewer algorithm iterations and hopefully better results.

1.3 Brief related work

Details and sources on our approach are given in the following section on algorithm development, and as such we give a brief overview of related work and literature in

this section. Clustering on graphs has received a lot of attention in recent years due to its applications in using it for community detection in network graphs, and there are extensive reviews available (such as Fortunato (2010) or Yen et al. (2008). In itself it is based on the classic, distance-based k-means (see for example Kaufmann and Rousseeuw (1990)).

In clustering tasks on graph the idea behind the distance-based k-means approach can be used as well, albeit on the data transformed to a graph and using kernels. This also adds a major advantage–as opposed to a distance-based k-means approach–kernel-based k-means can separate clusters that are non-linearly separable in input space (Dhillon et al., 2004). Furthermore, this approach enables the usage of different kernels for clustering and it can be easily expanded or adapted (Chitta et al., 2011).

1.4 Working Paper Structure

The rest of this working paper is structured as follows. In the following section, we give a brief overview of the existing literature on the k-means algorithm. We then proceed to develop the algorithm from the existing kernel k-means approach. In that section we also briefly present the Sum-over-Forests density index and the kernels we use for our experiments. The subsequent section covers our experiments, where we present the datasets, the parameters for kernels and algorithms as well as the results obtained. We conlude with a summary of our findings.

2 Algorithm, Density Index, Distances & Kernels

2.1 Algorithm – Generic Algorithm

We begin this section with a standard distance-based k-means-like clustering algorithm inspired in particular by the k-medoids method in Kaufmann and Rousseeuw (1990). It basically corresponds to a k-means-like two-steps iterative algorithm based on a distance (or dissimilarity) matrix, instead of features. Indeed, it is assumed that a symmetric distance matrix, $\mathbf{\Delta}$, containing the distances between every pair of nodes has been precomputed. The distance measure can be the shortest-path distance or any meaningful, local or global, distance (in our experimentation, we will use the free en-

ergy distance, the randomized-shortest-path dissimilarity–transformed to a distance– as well as the sigmoid commute time). The goal will be to partition the nodes by minimizing the total *within-cluster sum of distances*.

We have to stress that the standard k-means algorithm cannot be applied as-is on network data since it requires a data matrix and the computation of centroids in an Euclidean space, which are not available when working with networks. It is therefore useful to work around this issue by using either a distance matrix between nodes or a kernel on a graph, which implicitly defines a set of node vectors in an embedding space. In this working paper we exclusively employ the latter approach and transform the distances accordingly (see the experimental section for details).

For this algorithm, the number of clusters m has to be specified a priori by the user. Each cluster $\mathcal{C}_k$ (sometimes simply denoted as cluster k), with $k = 1, \ldots, m$, will be characterized by a *prototype* chosen among the nodes belonging to the cluster. This prototype is the "most typical" element of this cluster, and will be chosen as the most central node of the cluster: its distance to all the other nodes of the cluster is the smallest among all the nodes of the cluster. The variable containing the index of the node chosen as the prototype of cluster k will be denoted as q_k. For instance, q_k = 5 means that the prototype of cluster k is node number 5. The vector of prototype labels is $q = [q_1, q_2, \ldots, q_m]^T$.
We then define the *within-cluster sum-of-distance* of cluster k as

$$J_k = \sum_{j \in \mathcal{C}_k} \gamma_j \Delta_{q_k j} \tag{1}$$

corresponding to the sum of the distances of the nodes of cluster $\mathcal{C}_k$ to the prototype of the cluster. It quantifies the compactness of the cluster – a zero within-cluster sum-of-distances means that all elements of the cluster are superposed. Alternatively, we could consider the within-cluster inertia by computing the sum of *squared* distances instead, $J_k = \sum_{j \in \mathcal{C}_k} \gamma_j \Delta^2_{q_k j}$. This would amount to substitute the distance matrix by the matrix of squared distances $\mathbf{\Delta}^{(2)}$, where the superscript (2) indicates element-wise square of the matrix entries. Furthermore, we introduce the node weight γ and assume here that each node has a specific positive weight $\gamma_j > 0$. In our particular case we will make use of the Sum-Over-Forests density index (see the the following

section) to calculate weights. It is however feasible to choose weights using a different algorithm or methodology. If no weight is provided, by default, $\gamma_j = 1$.

The total within-cluster sum-of-distances is the sum of the within-cluster sum-of-distances,

$$J = \sum_{k=1}^{m} J_k = \sum_{k=1}^{m} \sum_{j \in \mathcal{C}_k} \gamma_j \Delta_{q_k j} \tag{2}$$

where the first sum is taken over the m clusters, while the second sum is taken on the nodes i belonging to cluster k, $i \in \mathcal{C}_k$. A low total sum-of-distances corresponds to a set of compact clusters. This criterion J should therefore be as low as possible.

Let us now introduce the binary membership values u_{ik}, which are equal to 1 if node i is assigned to cluster $\mathcal{C}_k$ and to 0 otherwise. The memberships u_{ik} are gathered in an $m \times n$ matrix $\mathbf{U}$. By using this membership matrix, J becomes

$$J = \sum_{j=1}^{n} \sum_{k=1}^{m} u_{jk} \gamma_j \Delta_{q_k j} \tag{3}$$

and our objective is to minimize this criterion with respect to the prototypes and the membership values. A block coordinate descend will be used to this end.

Re-computation of the prototype step. First, let us minimize the total sum-of-distances criterion with respect to the prototypes q_k while maintaining the memberships u_{ik}, that is the allocation of the nodes to the clusters, constant. From Equation (2), we observe that J is minimal when each q_k minimizes $\sum_{j \in \mathcal{C}_k} \gamma_j \Delta_{q_k j}$. Therefore, we must choose q_k such that

$$q_k \arg\min_{i \in \mathcal{C}_k} \left\{ \sum_{j \in \mathcal{C}_k} \gamma_j \Delta_{ij} \right\} \qquad \text{(recomputing the prototypes for each } k\text{)} \tag{4}$$

which aims to choose, as prototype, the node that is most central to the cluster k.

Re-allocation step. Then, we minimize J with respect to the binary membership values u_{ik} while keeping the prototypes constant. By looking to Equation (3), we see that $\gamma_j \sum_{k=1}^{m} u_{jk} \Delta_{q_k j}$ should be minimal for each j. Since $\sum_{k=1}^{m} u_{jk} = 1$ for all values

of j and the u_{jk} are binary, this is achieved by setting $u_{jk} = 1$ for the *smallest* $\Delta_{q_k j}$, and $u_{jk} = 0$ for the larger distances. In other words, node j should be assigned to the cluster k_j^* corresponding to the lowest $\Delta_{q_k j}$ – the *closest cluster*. Thus, the optimal cluster allocation k_j^* of node j is

$$k_j^* = \underset{k \in \{1,\dots,m\}}{\arg\min} \{\Delta_{q_k j}\} \qquad \text{(reallocation of the nodes for each } j\text{)} \tag{5}$$

and this choice decreases the criterion J. By denoting the cluster label of node j as $\ell(j)$, we have to set $\ell(j) = k_j^*$. The vector cluster label is ℓ.

Therefore, iterating these two steps decreases the criterion J at each iteration t by a non-negative amount $\delta J(t) = J(t-1) - J(t) \geq 0$. Since $J(t) > 0$ cannot become negative, $\sum_{t=1}^{\infty} \delta J(t) < J(0)$ so that $\delta J(t) \to 0$ and the objective function $J(t)$ converges to a local minimum. Notice, however, that the cluster assignment could oscillate if, for example, a node is equally distant to more than one prototype. But, in that case, the objective function remains stationary.

Pseudo-code for this algorithm is presented in Algorithm 1. As it depends on a random initialization of the prototypes, it is of common practice to launch several runs with different initializations and select the partition showing the lowest within-cluster sum-of-distances among all runs. We will explain our exact approach in the experimental section of this working paper. Notice also that the algorithm can also be implemented in matrix form, therefore avoiding expensive loops over the nodes.

2.2 Clustering with a kernel k-means

We now introduce a prototype-based kernel version of the simple k-means algorithm where the prototype vectors are defined in the sample space (see below). This clustering algorithm (Yen et al., 2008) assumes that a kernel matrix $\mathbf{K}$ containing similarities between the nodes of the graph G has been pre-computed. This kernel matrix should be positive semi-definite, which will be required for the theoretical development of the algorithm. However, in practice, it will only be required to be symmetric – indeed, it has been shown that the algorithm converges even if the similarity matrix is not positive semi-definite (see Yen et al. (2008) and discussion below). Thus, the technique is applicable to both directed and undirected weighted graphs, provided the

Algorithm 1 Standard distance-based k-means clustering algorithm.

Input:
- The $n \times n$ symmetric distance matrix
- The number of clusters m.

Output:
- The final $n \times m$ cluster membership matrix $\mathbf{U}$: $u_{ik} = 1$ if node i belongs to cluster k, zero otherwise.

1. **initialization**: randomly sample m different prototypes $[q_1, q_2, \ldots, q_m]^{\mathrm{T}}$ among the nodes. Here, the variable q_k contains the index of the node chosen as kth prototype.
2. $\mathbf{U} \leftarrow \mathbf{Zeros}(n, m)$ {$n \times m$ cluster membership matrix}
3. **repeat**
4. **for** $j = 1$ to n **do** {reallocation of nodes step}
5. $k_j^* \leftarrow \underset{k \in \{1,\ldots,m\}}{\arg\min} \{\Delta_{q_k j}\}$ {for each node j, find its closest prototype according to the distance}
6. $\ell(j) \leftarrow k_j^*$ {assign j to its closest cluster $\mathcal{C}_{k_j^*}$ and remove it from its previous cluster}
7. **end for**
8. **for** $k = 1$ to m **do** {recomputation of the prototype of each cluster}
9. $\mathcal{C}_k \leftarrow \{\text{node indexes } i : \ell(i) = k\}$ {gather nodes belonging to cluster k}
10. $q_k \leftarrow \underset{i \in \mathcal{C}_k}{\arg\min} \left\{ \sum_{j \in \mathcal{C}_k} \gamma_j \Delta_{ij} \right\}$ {as prototype of cluster $\mathcal{C}_k$, pick up the most central node, i.e. the node which is closest to the whole cluster $\mathcal{C}_k$}
11. **end for**
12. $J \leftarrow \sum_{k=1}^{m} \sum_{j \in \mathcal{C}_k} \gamma_j \Delta_{q_k j}$ {recompute the current value of the objective function J}
13. **until** convergence of the objective function J
14. **for** $i = 1$ to n **do** {fill in cluster membership matrix}
15. $u_{i\ell(i)} \leftarrow 1$
16. **end for**
17. **return** $\mathbf{U}$

kernel matrix computed on the graph is symmetric and, of course, captures similarities between nodes in a meaningful way.

A recent survey on kernel clustering together with its relationships with spectral clustering can be found in Filippone et al. (2008). Kernel k-means has been introduced, among others, by Dhillon et al. (2005); Du et al. (2005); Girolami (2002); Kim et al. (2005); Inokuchi and Miyamoto (2004); MacDonald and Fyfe (2000); Wu et al. (2003); Zhang and Chen (2004). Here, we will adopt a weighted, prototype-based kernel ver-

sion of the simple k-means algorithm where the prototype vectors are defined in the sample space, as detailed in Yen et al. (2008).

When dealing with kernel clustering, three different spaces are defined

- The **input space** is the initial space in which the data are defined. Recall that a nonlinear mapping is applied on this input space in order to obtain a new feature space (the embedding space) where the data are hopefully easier to handle (for instance, the classes are linearly separable) (Shawe-Taylor and Cristianini, 2004; Scholkopf and Smola, 2002). Each coordinate in the input space represents the measurement of a feature or characteristic on objects. This space is *not relevant* in our case since we are directly working on a graph structure, without having to explicitly define an input space.

- The **embedding space**, corresponding to the image of the input space through the mapping. In our case, the embedding space corresponds to the space in which the inner products between the node vectors of the graph, as provided by the kernel matrix, are computed. Thus, each node of the graph corresponds to a vector in this embedding space, called a **node vector**, and which can be computed by simple spectral decomposition of the kernel matrix and the theory of classical multidimensional scaling) But, as usual, these node vectors do not need to be computed; only the elements of the kernel matrix (the inner products between node vectors in the embedding space) are used in the clustering methods. The dimensionality of the embedding space corresponds to the rank of the kernel matrix.

- The empirical **sample space**, corresponding to the Euclidean space having, as dimensionality, the number of data samples. In the present case, this is n, the number of nodes in the graph. Each coordinate in this space thus corresponds to an object, that is, a node of the graph.

Basically, the kernel k-means relies on four steps:

1. First compute a meaningful *kernel* on the graph $\mathbf{K}$. This kernel induces an embedding space where each node is a vector and the similarity between two nodes i and j, k_{ij}, corresponds to the inner product between the two node

vectors in this space. Also, compute the density index for the graph, giving a vector γ of individual node weights.

2. Define a criterion, typically the sum of within-cluster inertia, depending on (1) **prototype vectors** in the embedding space (denoted as $\mathbf{g}_k$) and on (2) **membership values** indicating the membership of each node to the cluster (denoted as u_{ik}).

3. Express the prototype vectors in the embedding space in terms of the **prototype vectors in the sample space** (denoted as $\mathbf{h}_k$; a variant of the well-known kernel trick) in the criterion.

4. Optimize the criterion with respect to the prototype vectors, $\mathbf{h}_k$ using the density index γ as well as the membership values, u_{ik}. To this end, the standard two-step algorithm used in the k-means clustering and variants is applied: (i) optimize the prototype vectors while keeping the membership values fixed, and (ii) optimize the membership values while keeping the prototype vectors fixed. This algorithm is related to block coordinate descent.

This procedure is now applied in order to derive a kernel-based k-means clustering algorithm.

2.2.1 The kernel k-kmeans algorithm

Suppose we are looking for a partition in m clusters in total. The number of clusters is fixed a priori since the optimal value of the criterion on which k-means is based, the total within-cluster inertia, always decreases when the number of clusters increases. Thus, the goal is to design an iterative algorithm aiming to minimize a criterion – or objective function – which, in the case of a standard k-means, can be defined, in the embedding space, as the total **within-cluster inertia**:

$$J(\mathbf{g}_1, \ldots, \mathbf{g}_m) = \sum_{k=1}^{m} \sum_{i \in \mathcal{C}_k} \gamma_i \|\mathbf{x}_i - \mathbf{g}_k\|^2 \tag{6}$$

where the first sum is taken over the m clusters, while the second sum is taken on the nodes i belonging to cluster k, $i \in \mathcal{C}_k$. It is assumed that each node i has a corresponding positive weight $\gamma_i > 0$. In Equation (6), $\mathbf{x}_i$ is the **node vector** corresponding

to node i and $\mathbf{g}_k$ is a prototype vector of cluster k in the embedding space, while $\|\mathbf{x}_i - \mathbf{g}_k\|$ is the Euclidean distance between the node vector and the cluster prototype it belongs to. Therefore, the criterion defined in Equation (6) is the sum over all clusters of the within-cluster inertia of each cluster. This value should be as low as possible for a good clustering. The number of clusters, m, must be provided a priori by the user.

Here, the prototype vector $\mathbf{g}_k$ of one cluster is defined as a representative of this cluster k. Remember that the node vectors $\mathbf{x}_i$ represent the nodes in an Euclidean embedding space preserving exactly the inner products between nodes.

We denote by $\mathbf{X}$ the $n \times p$ (p is the number of features in the embedding space, and therefore its dimensionality) data matrix containing the transposed node vectors as rows; that is, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^{\mathrm{T}}$. Let us now define the following change of parameter:

$$\mathbf{g}_k \to \mathbf{X}^{\mathrm{T}}\mathbf{h}_k \tag{7}$$

corresponding to the so-called "kernel trick" (see, e.g., Bishop (2006); Scholkopf and Smola (2002); Shawe-Taylor and Cristianini (2004)). It aims to express the prototype vectors, $\mathbf{g}_k$, as a linear combination of the node vectors, $\mathbf{x}_i$ (the columns of $\mathbf{X}^{\mathrm{T}}$), $\mathbf{g}_k = \sum_{i=1}^{n} h_{ki}\mathbf{x}_i$. The $\mathbf{h}_k$ will be called the **prototype vectors** in the n-dimensional **sample space**.

Now, let us recompute the within-cluster inertia in terms of the $\mathbf{h}_k$ and the inner

products:

$$
\begin{aligned}
J(\mathbf{h}_1, \ldots, \mathbf{h}_m) &= \sum_{k=1}^{m} \sum_{i \in \mathcal{C}_k} \gamma_i (\mathbf{x}_i - \mathbf{g}_k)^{\mathrm{T}} (\mathbf{x}_i - \mathbf{g}_k) \\
&= \sum_{k=1}^{m} \sum_{i \in \mathcal{C}_k} \gamma_i (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_i - 2\mathbf{x}_i^{\mathrm{T}} \mathbf{g}_k + \mathbf{g}_k^{\mathrm{T}} \mathbf{g}_k) \\
&= \sum_{k=1}^{m} \sum_{i \in \mathcal{C}_k} \gamma_i (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_i - 2\mathbf{x}_i^{\mathrm{T}} \mathbf{X}^{\mathrm{T}} \mathbf{h}_k + (\mathbf{X}^{\mathrm{T}} \mathbf{h}_k)^{\mathrm{T}} \mathbf{X}^{\mathrm{T}} \mathbf{h}_k) \\
&= \sum_{k=1}^{m} \sum_{i \in \mathcal{C}_k} \gamma_i (k_{ii} - 2\mathbf{k}_i^{\mathrm{T}} \mathbf{h}_k + \mathbf{h}_k^{\mathrm{T}} \mathbf{K} \mathbf{h}_k) \\
&= \sum_{k=1}^{m} \sum_{i \in \mathcal{C}_k} \gamma_i (\mathbf{e}_i - \mathbf{h}_k)^{\mathrm{T}} \mathbf{K} (\mathbf{e}_i - \mathbf{h}_k) \qquad (8)
\end{aligned}
$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$, $k_{ii} = [\mathbf{K}]_{ii} = \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_i$, $\mathbf{k}_i = \mathbf{X}\mathbf{x}_i = \mathbf{col}_i(\mathbf{K}) = \mathbf{K}\mathbf{e}_i$.
By introducing the binary membership (indicator) matrix u_{ik}, the criterion J in Equation (8) can be rewritten as

$$
\boxed{J(\mathbf{h}_1, \ldots, \mathbf{h}_m) = \sum_{i=1}^{n} \sum_{k=1}^{m} u_{ik} \gamma_i (\mathbf{h}_k - \mathbf{e}_i)^{\mathrm{T}} \mathbf{K} (\mathbf{h}_k - \mathbf{e}_i)} \qquad (9)
$$

The membership value u_{ik} is equal to 1 if node i is currently assigned to cluster $\mathcal{C}_k$ and to 0 otherwise.

The k-means iteratively minimizes J by proceeding in two steps, (1) re-allocation of the node vectors (determination of the u_{ik}) while keeping the prototype vectors fixed, and (2) re-computation of the prototype vectors, $\mathbf{h}_k$, while maintaining the cluster labels of the nodes fixed.

Re-allocation step. Let us first consider that the prototype vectors are fixed. The binary membership values u_{ik} can be optimized independently for each node i in the sum of Equation (9) so that the best cluster allocation for a node i is to assign the node to the cluster for which $(\mathbf{h}_k - \mathbf{e}_i)^{\mathrm{T}} \mathbf{K} (\mathbf{h}_k - \mathbf{e}_i)$ is a minimum. Thus, clearly, the

re-allocation step minimizing J is

$$k_i^* = \underset{k\in\{1,\dots,m\}}{\arg\min} \left\{(\mathbf{h}_k - \mathbf{e}_i)^{\mathrm{T}}\mathbf{K}(\mathbf{h}_k - \mathbf{e}_i)\right\} \tag{10}$$

where the variable k_i^* contains the optimal cluster label of node i. Therefore, the elements of the ith row of the membership function become $u_{i,k_i^*} = 1$, and $u_{ik} = 0$ for $k \neq k_i^*$.

Re-computation of the prototypes. For the computation of the prototype vector, considering the cluster allocation as fixed, taking the gradient of J in Equation (8) with respect to $\mathbf{h}_k$ and setting the result equal to $\mathbf{0}$ yields

$$\mathbf{K}\mathbf{h}_k = \frac{\sum\limits_{i\in\mathcal{C}_k}\gamma_i\mathbf{K}\mathbf{e}_i}{\sum\limits_{i\in\mathcal{C}_k}\gamma_i} = \mathbf{K}\frac{1}{\eta_k}\sum_{i\in\mathcal{C}_k}\gamma_i\mathbf{e}_i \tag{11}$$

where $\eta_k = \sum_{i\in\mathcal{C}_k}\gamma_i$ is the total weight of nodes belonging to cluster $\mathcal{C}_k$. By looking carefully to Equation (11), we immediately observe from the left-hand side that $\mathbf{K}\mathbf{h}_k$ is a linear combination of the $\mathbf{k}_i = \mathbf{K}\mathbf{e}_i$. Therefore, one particular solution[1] to this system of linear equations is

$$\mathbf{h}_k = \frac{1}{\eta_k}\sum_{i\in\mathcal{C}_k}\gamma_i\mathbf{e}_i \tag{12}$$

In other words, $[\mathbf{h}_k]_i$ contains γ_i/η_k if node $i \in \mathcal{C}_k$ and 0 otherwise; it corresponds to a prototype vector defined for each cluster in the sample space. Therefore the *prototype re-computation step* is

$$[\mathbf{h}_k]_i = \begin{cases} \gamma_i/\eta_k & \text{if node } i \in \mathcal{C}_k \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

[1]The solution of Equation (11) is defined up to a vector lying in the null space of $\mathbf{K}$, which has no practical influence since the $\mathbf{h}_k$ are always multiplied by $\mathbf{K}$. We decide here to pick up the solution for which $\mathbf{h}_k^{\mathrm{T}}\mathbf{e} = 1$ and each $\mathbf{h}_k \geq 0$; that is, the elements of each column vector $\mathbf{h}_k$ are positive and sum to one.

This is a natural result since then $\mathbf{X}^{\mathrm{T}}\mathbf{h}_k$ corresponds exactly to the centroid of the cluster k in the embedding space (see Equation (7)). Therefore, the cluster prototype is the centroid of the cloud of node vectors in the embedding space, as for standard k-means.

This two-step procedure (Equations (10) and (13)) is iterated until convergence, and is very similar to the standard k-means algorithm. For a given cluster k, the prototype vector $\mathbf{h}_k$ contains the weighted degrees of membership of each node to cluster k. For each cluster, these degrees of membership are positive and sum to one.

The procedure necessarily converges to a local minimum of the criterion (6), even if the similarity matrix is not positive semi-definite. Indeed, each step (both re-allocation and prototype re-computation) decreases the criterion. Now, since it can be shown that it is bounded from below, the decrease must tend to zero, meaning that the criterion converges to a fixed value (see Yen et al. (2008) for details). Moreover, this generic methodology can easily be used in order to derive kernel versions of other standard clustering algorithms, such as fuzzy clustering or clustering by Gaussian mixtures, as shown in Yen et al. (2008).

Algorithm 2 A simple kernel k-means clustering of the nodes of G.

Input:

– A weighted directed or undirected graph G containing n nodes.

– $\mathbf{K}$: a $n \times n$ symmetric similarity matrix computed from G.

– The desired number of clusters, m.

Output:

– The $n \times m$ membership matrix $\mathbf{U}$ containing the membership of each node i to cluster k, u_{ik}.

1. **initialization**: choose m different prototype nodes with indices $q_1, q_2, \cdots, q_m$ at random and set $\mathbf{h}_k = \mathbf{e}_{q_k}, k = 1 \cdots m$, where $\mathbf{e}_i$ is a $n \times 1$ basis column vector (contains 0's everywhere except at position i where it contains a 1).
2. **repeat**
3. $\quad \mathbf{U} \leftarrow \mathbf{Zeros}(n, m)$
4. $\quad$ **for** $i = 1$ to n **do**
5. $\quad\quad k^* \leftarrow \underset{k \in \{1,\ldots,m\}}{\arg\min} \left\{ (\mathbf{h}_k - \mathbf{e}_i)^{\mathrm{T}} \mathbf{K} (\mathbf{h}_k - \mathbf{e}_i) \right\}$ {re-allocation step}
6. $\quad\quad u_{ik^*} \leftarrow 1$ {store the cluster allocations}
7. $\quad$ **end for**
8. $\quad$ **for** $k = 1$ to m **do**
9. $\quad\quad \eta_k \leftarrow \sum_{i=1}^{n} \gamma_i u_{ik}$ {compute the total weight of nodes in each cluster}
10. $\quad\quad \mathbf{h}_k \leftarrow \dfrac{\boldsymbol{\gamma} \circ \mathbf{col}_k(\mathbf{U})}{\eta_k}$ {re-compute the prototypes}
11. $\quad$ **end for**
12. **until** the allocation of the nodes does not change any more
13. **return** $\mathbf{U}$

2.3 Density Index

In this section we will briefly present the density index used for our experiments. In particular, we use the Sum-over-Forests density index (Senelle et al., 2014). The Sum-over-Forests density index is derived from the Boltzmann probability distribution for a graph's rooted forest using free energy. For the sake of brevity, we will not go into the development of the index and instead refer the interested reader to Senelle et al. (2014). However, we will present the weighted and unweighted versions of the Sum-over-Forests density index, both of which we use in our experiments.

We define the following matrices:

$$\mathbf{W} = \exp\left(-\theta \mathbf{C}\right), \tag{14}$$

$$\mathbf{Z} = \left(\mathbf{I} + \mathbf{L}(\mathbf{W})\right)^{-1} = \left(\mathbf{I} + \left(\mathbf{diag}(\mathbf{W}^T\mathbf{e}) - \mathbf{W}\right)\right)^{-1} \tag{15}$$

where $\mathbf{C}$ is the graph's cost matrix, θ is a parameter, $\mathbf{I}$ is the identity matrix, $\mathbf{L}$ is the Laplacian matrix and $\mathbf{e}$ is a basis column vector. Using these two equations we can proceed to calculate the density index as follows.

Unweighted Density Index

The unweighted density index is identical to the formulation in Senelle et al. (2014):

$$\mathbf{d} = \mathbf{W}\,\mathbf{diag}(\mathbf{Z}) - \mathbf{diag}(\mathbf{WZ}). \tag{16}$$

Weighted Density Index

The weighted density index is similar but slightly different:

$$\mathbf{d} = \mathbf{diag}(\mathbf{Z})(\mathbf{W}^T\mathbf{e}) - \mathbf{diag}(\mathbf{WZ}). \tag{17}$$

2.4 Distances & Kernels

In this section we will briefly present the distances and kernels that we use to run our experiments. Please note that the distances and dissimilarities are transformed into a

kernel $\mathbf{K}$ using the following equation:

$$\mathbf{K} = -\frac{1}{2}\,\mathbf{H}\,\mathbf{D}^2\,\mathbf{H} \tag{18}$$

where $\mathbf{D}$ is a square distance or dissimilarity matrix and $\mathbf{H} = \mathbf{I} - \frac{1}{n}$, where n is the number of nodes. The exponent is fixed to a value of 2 as pretests have shown little influence and good results with this value. However, other values are feasible also.

2.4.1 Free Energy Distance

Kivimäki et al. (2014) proposed to base a distance on the computation of the Helmholtz **free energy** between two graph nodes i and j, which results in the **free energy** to be defined as:

$$\Delta_{ij}^{\phi} = \frac{\phi(i,j) + \phi(j,i)}{2} \quad \text{with} \quad \phi(i,j) = -\frac{1}{\theta}\log\frac{z_{ij}}{z_{jj}} \quad \text{where} \quad i \neq j \tag{19}$$

We will refer to the derived kernel with the acronym FE.

2.4.2 Randomized Shortest Path Dissimilarity

Kivimäki et al. (2014) equally proposed the randomized shortest path dissimilarity between two graph nodes i and j, which is the average expected cost for going from i to j and then back to i. We refer the reader to Kivimäki et al. (2014) for the details. We will refer to the derived kernel with the acronym RSP.

2.4.3 Sigmoid Commute Time Kernel

Yen et al. (2007) developed a sigmoid commute-time kernel that is based on the average commute time as average number of steps a random walker will take, when starting in node $i \neq j$, going to j and back to i. We will refer to the derived kernel with the acronym SCT.

3 Experiments

3.1 Datasets

3.1.1 Newsgroups

The newsgroups dataset is a collection of circa 20000 unstructured newsgroup documents taken from 20 discussion groups on UseNet. The data is spread across 20 different groups. Since its original inception the dataset has become popular for experiments in machine learning. For our experiments we use preprocessed data from Yen et al. (2008), that is present in graph from and refer to said article for details on the preprocessing. The different sets vary in size and contain 400, 600 or 1000 nodes.

3.1.2 Zachary's Karate Club

Zachary's karate club is social network of friendships between 34 members of a karate club at a US university in the 1970s (Zachary, 1977). The club is divided into two different groups and our goal is to cluster the members correctly into their respective groups.

3.1.3 Political Blogs

Political blogs is a directed network of hyperlinks between weblogs on US politics, recorded in 2005 by Adamic and Glance (2005). The 105 blogs are classified into three groups according to their political orientation: conservative, liberal or neutral; this classification should be found by an algorithm.

3.1.4 LFR

The LFR datasets consist of three graphs computed according to Lancichinetti et al. (2008). In particular, the datasets are made up of binary networks with overlapping nodes, thus increasing clustering difficulty. All three graphs are comprised of 600 edges each.

3.2 Parameters & Parameter Tuning

In addition to ascertain the feasability of using a density index-based weighted kernel k-means approach, the goal of our research is to analyze the role and impact of the various parameters in the different steps of the algorithms and subroutines that make up the entirety of our clustering methodology.

Table 1 gives an overview of the different parameters available and their values tested.

Name	**Tested Range**
Sum-over-Trees θ	$0.005 \ldots 100$
Density evaluation type	none, max, 2norm, smooth
FE β	$0.001 \ldots 100$
SCT α	$0.007 \ldots 700$
RSP β	$0.001 \ldots 100$

Table 1: Overview of parameter values tested

The first parameter, the Sum-over-Trees θ, is a biasing parameter that gradually shifts the probabilities of forests employed in the density index calculation towards low-cost forests. We will skip the theoretical bits and instead describe its effects; the higher the value of θ, the more extreme the density values. The density evaluation determines the method by which the a node's density is judged in relation to its degree; the difference between the four options is negligible. We therefore omit its explicit analysis here. Finally, the kernel parameters α and β influence the clustering quality as well; generally higher values give better results, but there is a more or less optimal value that has to be found by testing the parameters.

3.3 Statistical measures

3.3.1 Normalized Mutual Information (NMI)

The Normalized Mutual Information (NMI) is a measure of agreement between the actual clustering and the clustering found by an algorithm. The NMI ignores permutations and indicates to which level the found clustering and the actual clustering

share the same information. In case of a perfect correlation between the two the NMI score is equal to 1.

3.3.2 Adjusted Rand Index (ARI)

The Adjusted Rand Index (ARI) is a corrected-for-chance version of the Rand index and is a measure of the comparison of partitions. As such, we use it to compare the found clustering with the actual clustering of the data. We refer the reader to Hubert and Arabie (1985) for details on its calculation.

3.3.3 Classification Rate (CR), Micro & Macro Precision & Recall

The classification rate returns the percentage of correct classification. Precision and Recall indicate the average precision or positive predictive value, that is how many elements were correctly assigned to a cluster out of all elements that were assigned to that cluster by the algorithm as well as the average recall or sensitivity, which indicates how many elements were correctly assigned to a cluster out of all elements present in the original cluster. We calculate the averages for these measures using both a micro- and macro-average approach.

3.3.4 Runtime

We provide the average runtime for the individual trials (see Section (3.4 for a our definition of a trial), which is given in seconds (or milliseconds). Please note that these values should be compared only in-between individual experiments themselves as different servers with different CPUs were used to run the experiments and thus the runtime should be considered only as an indication. The runtime strongly depends on the dataset's size and on the employed clustering method. However, the former has a much stronger influence than the latter.

3.3.5 Iterations

The number of iterations until the algorithm reaches convergence, that is the assignment of indivdiual nodes to a cluster centroid changes insignificantly or not at all anymore is a measure of comparison between the different clustering methods.

3.4 Experimental setup

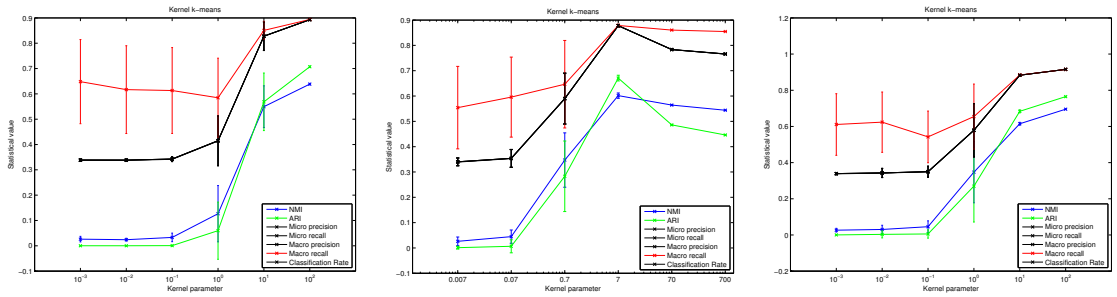
With the different kernels and parameters described above we ran an experimental schedule as follows. We tested combinations of all parameters and kernels where the parameters were varied by logarithmic steps, that is magnitudes apart. Furthermore, we ran a standard kernel k-means, a weighted kernel k-means with (i) unweighted and (ii) weighted density index in parallel with the same initial centroids. For each parameter combination we used the average of 50 runs. A run consists of 50 trials, where only the single best result of these 50 trials is kept. Each trial sees new random initial centroids, but they remain the same for each method.

3.5 Results

In this section we will show some preliminary results of our comparison.

3.5.1 Kernel parameter tuning

In this section we will look at the optimal kernel parameters for the sigmoid commute time kernel (SCT), the free energy kernel (FE) and the randomized shortest path kernel (RSP). In order to find the optimal values we use a specialized dataset from the news datasets, here termed the parameter tuning news dataset.



(a) FE kernel (b) SCT kernel (c) RSP kernel

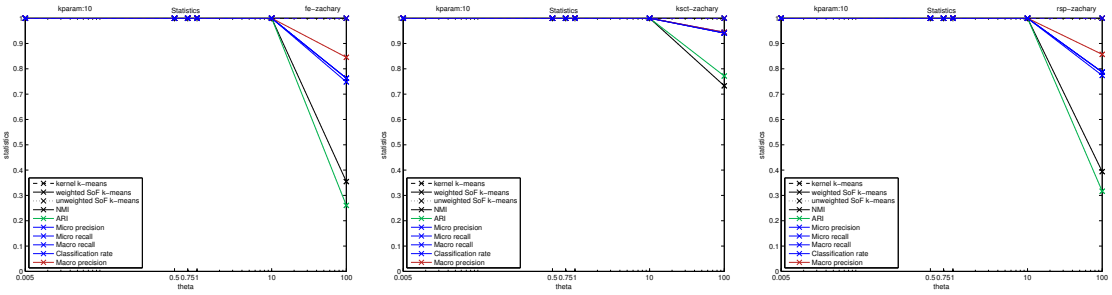
Figure 2: Optimal kernel choice on specialized dataset

Even though the FE kernel and the RSP kernel showed better results with an even higher value of β, the runtime doubled in the step from $\beta = 10$ to $\beta = 100$. With larger datasets and many parameter values to test, the higher runtime becomes an issue, which is why we go with the recommendation of the developers of the FE

and RSP distances to limit the value of β to a maximum of 20 for testing and 10 for running most of our experiments. For the SCT kernel we go with the parameter value $\alpha = 10$, as further in-detail testing showed that it gives slightly better results than the indicated value of $\alpha = 7$ in Figure 2b.

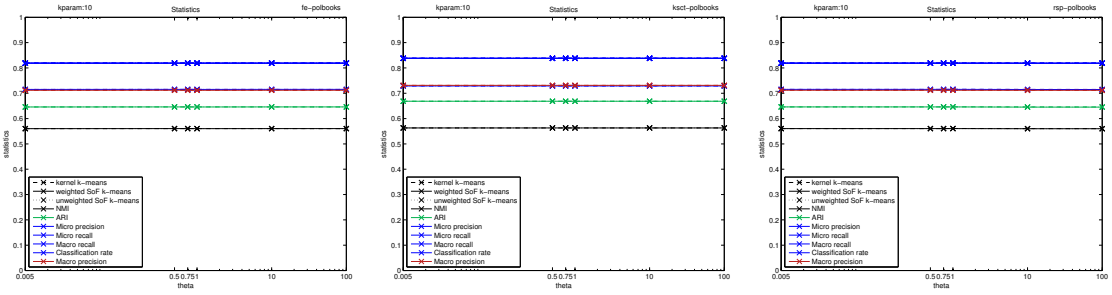
3.5.2 Kernel k-means vs. weighted kernel k-means

In this section we will compare the existing kernel k-means approach against the weighted kernel k-means approach. The latter will be based on a weighted density index as well as an unweighted density index. Also, we will do the comparisons using the different kernels presented with their respectively tuned parameters.



(a) FE kernel (b) SCT kernel (c) RSP kernel

Figure 3: Zachary Karate Club statistical measure results



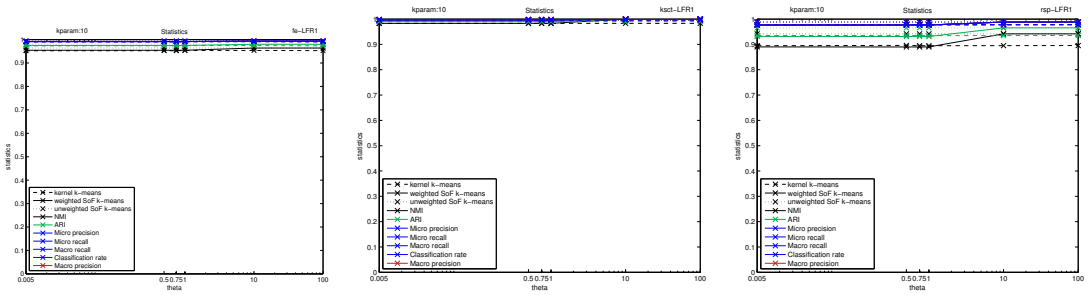
(a) FE kernel (b) SCT kernel (c) RSP kernel

Figure 4: Political blogs statistical measure results

The smaller Political blogs and Zacharay datasets were solved by all methods and parameter values of almost identically. Only for higher values of θ the Zachary dataset

shows that the statistical measures drop. This drop is not present in the Political blogs dataset results. The difference between the kernels is only discernible at the highest value of θ for the Zachary dataset. Here the SCT kernel performs better than the other two. Similarly, the SCT kernel shows slightly better results than the FE and RSP kernels for the political blogs dataset, both of which give nearly identical results.

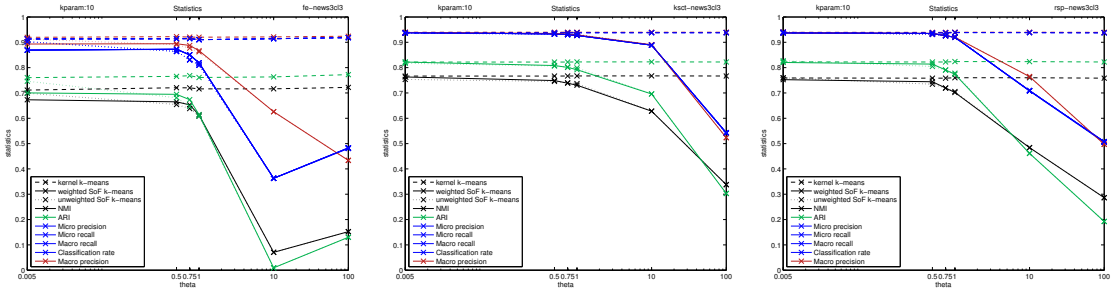
The LFR datasets showed close to identical results for all methods. We therefore present only one set of results for the LFR datasets in Figure 5.



(a) FE kernel (b) SCT kernel (c) RSP kernel

Figure 5: LFR1 statistical measure results

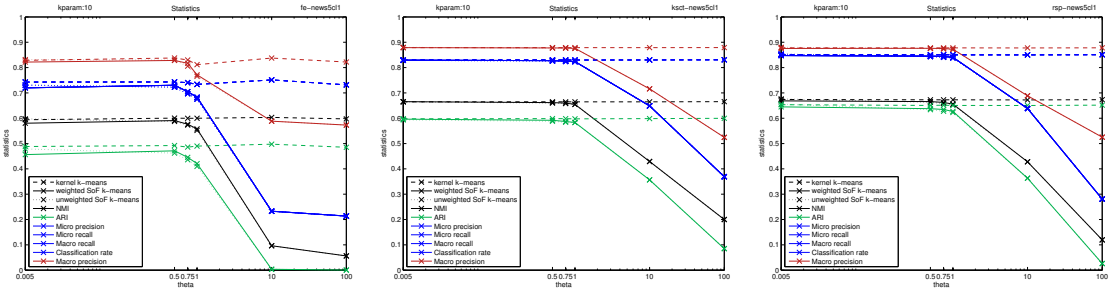
The aforementioned tendency for stastical measures to drop is not present in any of the LFR dataset results. Indeed, the results are all again very similar, with the SCT kernel slightly outperforming the FE kernel, which in turn outperforms the RSP kernel.



(a) FE kernel (b) SCT kernel (c) RSP kernel

Figure 6: news3cl3 statistical measure results

The same does not hold true for the news datasets. We present two three class datasets in the following Figures 6 and 7. As it can clearly be seen the the results start dropping as soon as the density index parameter reaches any sort of remarkable value ($\theta \approx 0.5 \ldots 1$) and go into free fall from then on to the point where even guessing would give better results on average. While the SCT kernel seems to handle the higher value of θ somewhat better, the other two kernels rapidly give bad results. The SCT kernel gives very similar results to the RSP kernel and both outperform the FE kernel.

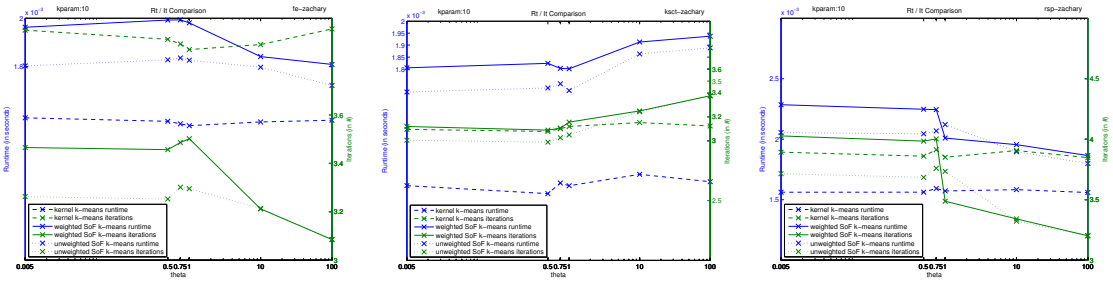


(a) FE kernel (b) SCT kernel (c) RSP kernel

Figure 7: news5cl1 statistical measure results

3.5.3 Runtime & Iterations

Two measures are not apparent from the previous section: runtime and iterations. In this section we will look at how the clustering methods and kernels behave in terms of algorithm runtime and the number of iterations until the clustering is deemed complete.

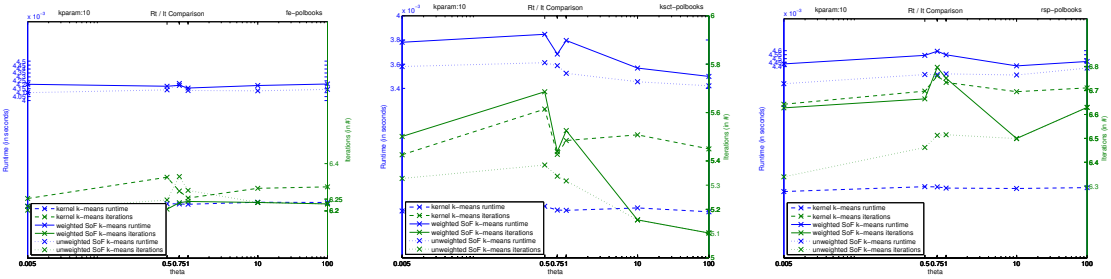


(a) FE kernel (b) SCT kernel (c) RSP kernel

Figure 8: Zachary Karate Club runtime and iterations results

Figures 8 and 9 show the runtime and iterations–on the left and right axis respecitvely– for the Zachary and Political blogs datasets. While for the Zachary dataset the number of iterations and runtime clearly decrease for the FE and RSP kernels, it stagnates or even rises for the SCT kernel. Also, one has to keep in mind that for high values of θ the statistical measures dropped for high values of θ.

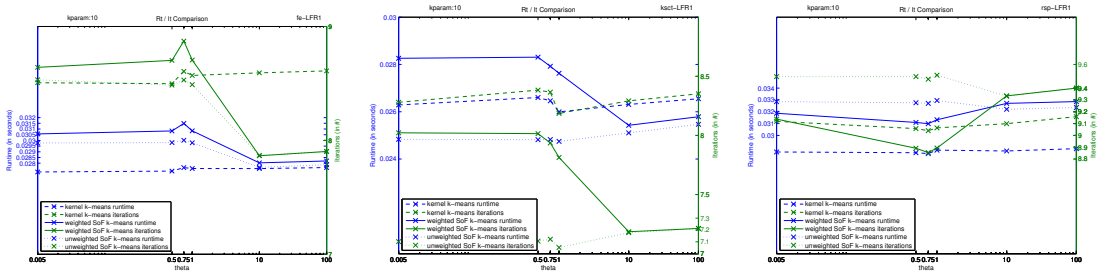
Conversely, for the Political blogs dataset, the SCT kernel saw a drop in number of iterations and runtime, while the RSP and FE kernels gave somewhat constant values.



(a) FE kernel (b) SCT kernel (c) RSP kernel

Figure 9: Political blogs runtime and iterations results

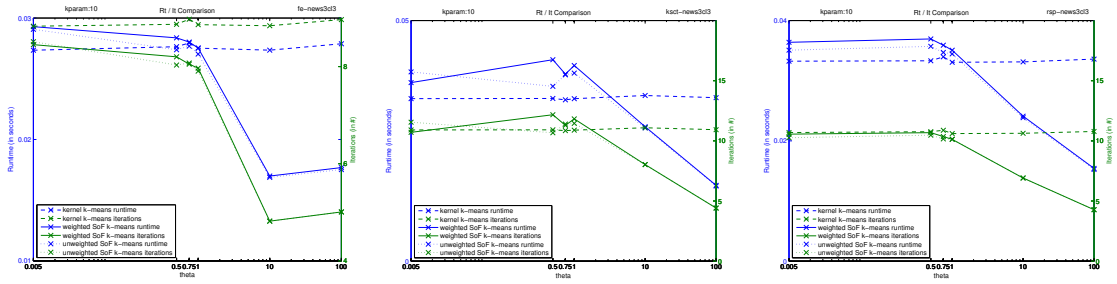
The LFR dataset showed similar behavior with no clear trend as to a strongly correlated reduction in runtime and iterations.



(a) FE kernel (b) SCT kernel (c) RSP kernel

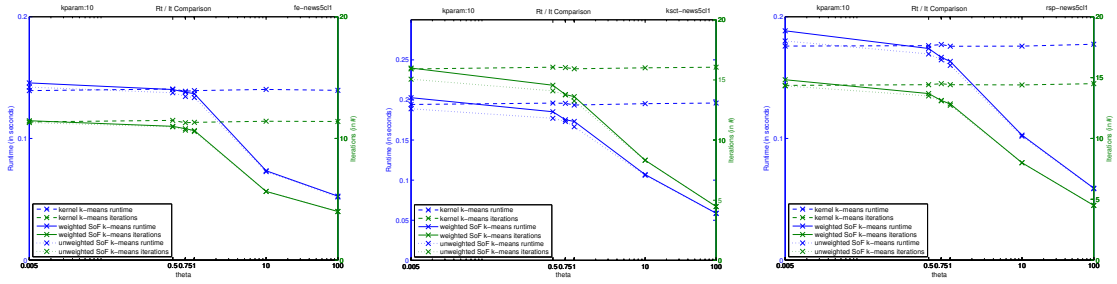
Figure 10: Political blogs runtime and iterations results

However, the news datasets showed a very clear correlation as can be seen in Figures 11 and 12. Here, the runtime and number of iterations drop as the values of θ go up. This holds true for both news datasets and for all kernels. Keep in mind though, that the statistical measures unfortunately showed similar behavior for higher values of θ.



(a) FE kernel (b) SCT kernel (c) RSP kernel

Figure 11: news3cl3 statistical measure results



(a) FE kernel (b) SCT kernel (c) RSP kernel

Figure 12: news5cl1 statistical measure results

3.6 Discussion

The developed weighted k-means algorithm with weights based on density clearly works, but is strongly dependant on the density index. Using the weighted density index vs. the unweighted density index neither has a signicant effect on the statistical result scores nor on the runtime and iteration count. However, the density index parameter θ has a tremendous effect on the results. Low values of θ result in nearly the same results for the clustering, both in terms of statistical measures as well as runtime and iteration count, while high values of θ lead to both lower scores for statistical measures as well as lower runtime and lower iteration count. These facts beg the question, if a middle ground can be found, wherein a small sacrifice in statistical measure score give (possibly more important) faster runtime and lower iteration count, especially given the fact that the developed algorithm manages to achieve faster runtimes and lower iteration counts towards convergence than the standard kernel k-means based algorithm. A more detailed analysis on the results, in particular using statistical tools

is necessary however to tell apart the variations from the actual results, to discern between signal and noise so to speak.

Furthermore, there is more work to be done on the details. As such, each kernels' parameter could also be tuned for each dataset to allow for better individual results on the dataset level. Testing other kernels and comparing to other clustering methods is also of interest. In addition, it could be interesting to use other density indices as weights in the algorithm.

4 Summary

In this work, we developed a weighted kernel k-means approach, based on the Sum-over-Forests density index. Furthermore, we implemented the algorithm and tested various parameters on both the kernels and the density index and compared our approach with a standard kernel k-means based method. While the algorithm clearly works and is on-par with the existing kernel k-means approach, it has to sacrifice statistical measure score in order to show a significant improvement on runtime and iteration count.

5 Acknowlegdements

We would like to thank Pascal Francq, Bertrand Lebichot and Fabian Lecron for their valuable comments and insights. This article is supported in part by the FNRS through a PhD scholarship.

References

Adamic, L. A. and Glance, N. (2005). The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM.

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer.

Chitta, R., Jin, R., Havens, T. C., and Jain, A. K. (2011). Approximate kernel k-means: Solution to large scale kernel clustering. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 895–903. ACM.

Dhillon, I. S., Guan, Y., and Kulis, B. (2004). Kernel k-means, spectral clustering and normalized cuts. In *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 551–556. ACM Press.

Dhillon, I. S., Guan, Y., and Kulis, B. (2005). A unified view of kernel k-means, spectral clustering and graph cuts. In *UTCS Technical Report TR-04-25*.

Du, W., Inoue, K., and Urahama, K. (2005). Robust kernel fuzzy clustering. In *Proceedings of Fuzzy Systems and Knowledge Discovery (FSKD 2005)*, Lecture Notes in Artificial Intelligence 3613, pages 454–461.

Filippone, M., Camastra, F., Masulli, F., and Rovetta, S. (2008). A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41:176–190.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174.

Girolami, M. (May 2002). Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, 13(3):780–784.

Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2(1):193–218.

Inokuchi, R. and Miyamoto, S. (2004). Lvq clustering and som using a kernel function. *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 1497–1500.

Kaufmann, L. and Rousseeuw, P. (1990). *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons.

Kim, D.-W., Lee, K. Y., Lee, D., and Lee, K. H. (2005). Evaluation of the performance of clustering algorithms in kernel-induced feature space. *Pattern Recognition*, 38(4):607–611.

Kivimäki, I., Lebichot, B., and Saerens, M. (2014). Developments in the theory of randomized shortest paths with a comparison of graph node distances. *Physica A: Statistical Mechanics and its Applications*, 393:60016.

Lancichinetti, A., Fortunato, S., and Radicchi, F. (2008). Benchmark graphs for testing community detection algorithms. *Physical review E*, 78(4):46–110.

MacDonald, D. and Fyfe, C. (2000). The kernel self organising map. *Proceedings of the fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, pages 317–320.

Scholkopf, B. and Smola, A. (2002). *Learning with kernels*. The MIT Press.

Senelle, M., Garcia-Diez, S., Mantrach, A., Shimbo, M., Saerens, M., and Fouss, F. (2013). The sum-over-forests density index: Identifying dense regions in a graph. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PP(99):1–1.

Senelle, M., Garcia-Diez, S., Mantrach, A., Shimbo, M., Saerens, M., and Fouss, F. (2014). The sum-over-forests density index: identifying dense regions in a graph. *ArXiv:1301.0725; to appear in the IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(6):1268–1274.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge University Press.

Wu, Z.-D., Xie, W.-X., and Yu, J.-P. (2003). Fuzzy c-means clustering algorithm based on kernel method. In *ICCIMA '03: Proceedings of the 5th International Conference on Computational Intelligence and Multimedia Applications*, page 49, Washington, DC, USA. IEEE Computer Society.

Yen, L., Fouss, F., Decaestecker, C., Francq, P., and Saerens, M. (2007). Graph nodes clustering based on the commute-time kernel. In *Proceedings of the 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2007)*, Lecture Notes in Artificial Intelligence 4426, pages 1037–1045.

Yen, L., Fouss, F., Decaestecker, C., Francq, P., and Saerens, M. (2008). Graph nodes clustering with the sigmoid commute-time kernel: A comprehensive study. *Data & Knowledge Engineering*, 68(3):338–361.

Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, (33):452–473.

Zhang, D.-Q. and Chen, S.-C. (2004). A novel kernelized fuzzy c-means algorithm with application in medical image segmentation. *Artificial Intelligence in Medicine*, 32(1):37–50.