Kernel K-means

Javier Ferrando, Guillermo Lahuerta, Marcel Porta

December 2019

1 Introduction

Clustering is the task of dividing data points into homogeneous groups. In other words, is the process of separating points with similarities and assign them to clusters. The need for clustering algorithms appears in different fields, such as data mining, image processing or pattern matching, among others.

There are multiple algorithms that perform this task. However, they significantly differ in the way that they consider that two given data points are "similar" or not (e.g, centroid-based methods, density-based methods, distribution-based methods).

Among all the algorithms available in the literature, one of the most popular is k-means which is a centroid-based technique.

2 K-means

The main reason why k-means is so popular, is its simplicity. We first initialize the centroid of k clusters and, at each iteration, we minimize the sum of the squared Euclidean distances between each data point and its corresponding cluster. We update the centroid of each cluster and repeat this process until convergence.

However, this simple technique suffers from serious disadvantages:

- The number of clusters k must be defined by the user.
- The solution strongly depends on the initialization of the centroids of the clusters.
- It can only find linearly separable clusters.

Different options exists to deal with these issues. In this paper, we will focus on a popular technique to modify the standard k-means in a way that it is capable to find nonlinearly separable clusters, the *kernel k-means*.

3 Previous work

The idea of modifying a k-means to make it suitable for non-linearly separable clusters, was first described in [1], back in 1998. However, Kernel-based algorithms have been studied in many other problems, especially in the field of Support Vector Machines [4].

4 Kernel K-means

When clusters are not linearly separable, the k-means clustering algorithm can be enhanced by the use of a kernel function; by using an appropriate nonlinear mapping function ϕ, we map the data points into a higher dimensional (feature) space, where clusters are linearly separable and, therefore, we can apply k-means. This results in linear separators in feature space which corresponds to nonlinear separators in input space. With this technique, we avoid the limitations of linearly separable clusters in input space that k-means suffers from.

The goal of kernel k-means is minimize the clustering error in feature space. We can define a *kernel* matrix $K \in R^{NxN}$, where $K_{i,j} = \phi(X_i)^T\phi(X_j)$ and by taking the advantage of the kernel trick we can compute the squared Euclidean distances without explicit knowledge of the ϕ function (how this occurs is explained later). Any positive-semidefinite matrix can be used as a kernel matrix. Important difference now is in the centroids $\mathbf{c}_k$ (center points of the cluster) cannot be calculated in the feature space. Usually a kernel function does not provide the coordinates of the data in the feature space, this directly provides the inner products in feature space. In the Table 3 there are some of the most popular Kernel functions.

Table 1: Kernel functions

Polynomial Kernel	$K(x_i, x_j) = (x_i^T x_j + \gamma)$
RBF Kernel	$K(x_i, x_j) = \exp$ (- $\|\|(x_i - x_j)\|\|^2/2\sigma^2$)
Sigmoid Kernel	$K(x_i, x_j) = \tanh(\gamma(x_i^T x_j + \theta)$

Kernel k-Means algorithm

As has been explained previously the goal is to minimize the clustering error, which is expressed in the following way with the squared euclidean distances between the points and the centroids:

$$E(c_1, .., c_k) = \sum_{n=1}^{N}\sum_{k=1}^{M} I(x_n \in C_k)||\phi(X_n) - c_k||^2 \tag{1}$$

- $I(x_n \in C_k)$: 1 if x_n is in the cluster k, 0 otherwise.
- c_k is the set of k centroids.

Recall that previously we said that to obtain the cluster centroid in feature space it is not possible. However, we can write the center point as a function of all the points that belongs to one cluster:

$$c_k = \frac{\sum_{n=1}^{N} I(x_n \in C_k)\phi(X_n)}{\sum_{n=1}^{N} I(x_n \in C_k)} \tag{2}$$

Note that in the previous equation, it appears the mapping function $\phi(\cdot)$, which is unknown. Therefore, we have to change it to Kernel evaluations. Finally, we have:

$$||\phi(X_n) - c_k||^2 = K_{nn} - \frac{\sum_{n=1}^{N} I(x_n \in C_k)K_{nj}}{\sum_{j=1}^{n} I(x_n \in C_k)} + \frac{\sum_{j=1}^{N}\sum_{l=1}^{N} I(x_j \in C_k)I(x_l \in C_k)K_{jl}}{\sum_{j=1}^{N}\sum_{l=1}^{N} I(x_j \in C_k)I(x_l \in C_k)} \tag{3}$$

- K is the resulting kernel matrix, obtained after applying some of the kernel functions in table 3.

With these equations the algorithm is the following:

- **Input**: Kernel matrix K, number of clusters k and an initial matrix of clusters. The matrix of clusters is a matrix with as much rows as points and as columns as clusters; if one element belongs to cluster one, this element will have a one in the first column.
- **Output**: Matrix clusters, where all elements belongs to one cluster.

Algorithm .1: Kernel k-means

```
for all points x_n, n = 1,...N do
    for all clusters C_i, i = 1 to k do
        Compute ||φ(x_n) − c_i||^2 using the equation (3)
    end for
    find c*(x_n) = argmin(||φ(x_n) − c_i||^2)
end for
for all clusters C_i, i = 1 to k do
    Update cluster C_i = {x_n|c*(x_n) = i}
end for
if converged then
    return final clusters C_i, i = 1 to k
else
    Go to step 1
end if
```

Kernel k-Means R implementation

In this project we have implemented the previous algorithm, and in this section we will explain some pieces of code with the aim to clarify how it has been done.

- **The main algorithm**

```
kkmeans_own <- function(K, c, clusters, N){
  new.clusters <- matrix(nrow=N, ncol=c) # Matrix of new clusters
  for (n in 1:N){
    distances <- c()
    for (i in 1:c){
        # Call to get_distances function that implements equation 2
      distances <- c(distances, get_distance(K, n, i, clusters))
    }
    # Update cluster matrix with the cluster that has obtained the min value
    new.clusters[n,] <- as.numeric(distances == min(distances))
  }
  # If it converges
  if(identical(new.clusters, clusters)){
    return(new.clusters) # End return the result
  }else{
    kkmeans_own(K, c, new.clusters, N) # Recursive call
  }
}
```

- Implementation of the third equation (3). This step is the most important because it is the computation of the distances of each point X_n to the k clusters. We use a

matrix notation to avoid for loops:

$$K_{nn} - \frac{\sum_{n=1}^{N} I(x_n \in C_k) K_{nj}}{\sum_{j=1}^{n} I(x_n \in C_k)}$$

is equal to:

```
K[n,n] - 2*I[,k] %*% K[,n] / sum(I[,k])
```

then,

$$\frac{\sum_{j=1}^{N} \sum_{l=1}^{N} I(x_j \in C_k) I(x_l \in C_k) K_{jl}}{\sum_{j=1}^{N} \sum_{l=1}^{N} I(x_j \in C_k) I(x_l \in C_k)}$$

is equal to:

```
sum(I[,k] * K %*% I[,k]) / (I[,k]%*%I[,k])**2
```

So, adding the two parts, we obtain for one point x_n, the distance to all the clusters k, looking in all the elements that belongs to this cluster.

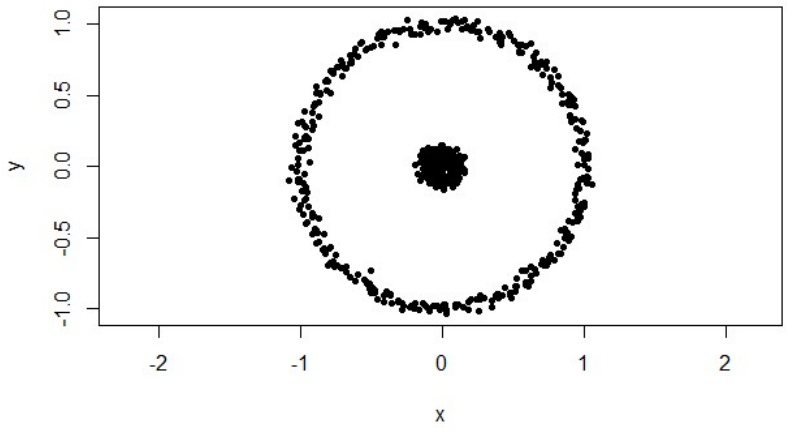
5 Experimental evaluation

In this section we test the performance of the kernel k-means. For that, we use three different algorithms (the standard k-means from library 'kernlab', the kernelized k-means also from library 'kernlab', and our implementation of the kernel k-means) against two benchmark datasets. This will allow us to compare the performance, not only between the standard and the kernelized version of k-means, but also between our own implementation and the one contained in kernlab, one of the most popular libraries in R for kernel algorithms.

Rings dataset

The first dataset used to compare the performance of these methods consists of a small circle of points, surrounded by a ring. This 2-D dataset is presented in Figure 1.

Figure 1: Rings dataset.

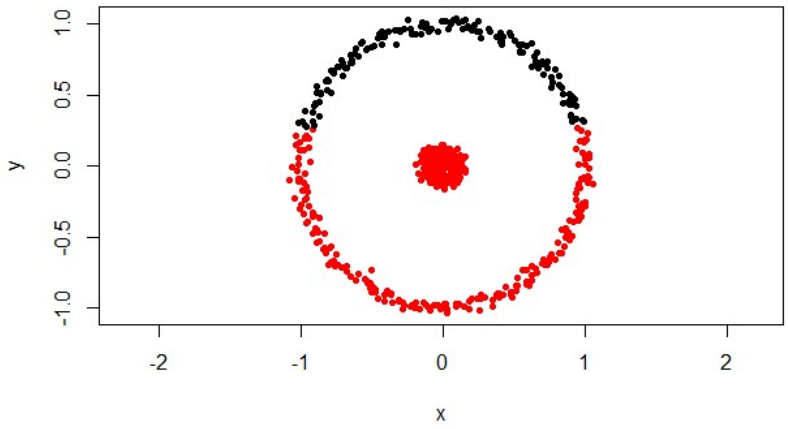


Note that this synthetic dataset has been generated in a way that the data is not linearly separable. The expected outcome is that the standard k-means will fail to cluster properly

the data points, while the kernelized version, if the kernel function is correctly selected, will success.

The first method to be used to cluster the synthetic data is the classical k-means algorithm, already implemented in library 'kernlab'. To use this function, it is as simple as call the function *kmeans()* and pass as attributes the matrix of coordinates of the data points and the number of clusters k. The clustering obtained is presented below:

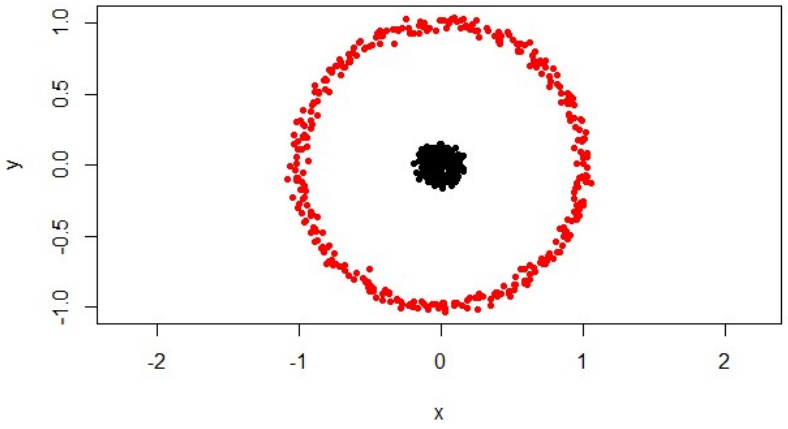
Figure 2: Rings dataset k-means clustering.



As expected, since the clusters are not linearly separable, the classical k-means fails to cluster the two groups of data points.

The next algorithm to be tested is the kernel k-means that is already implemented in the package 'kernlab'. To use this method, one must call function *kkmeans()* and pass as arguments the matrix of coordinates of the data points, the number of clusters, and the kernel function chosen. In this case we use the Radial Basis kernel Function (RBF). The clustering obtained is presented in the following figure:

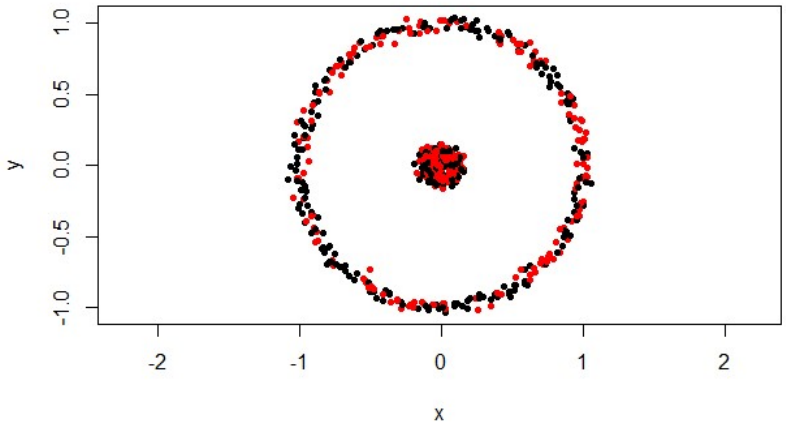
Figure 3: Rings dataset kernel k-means clustering.



We can see that the kernelized version is perfectly capable to cluster the data points in two correct clusters.

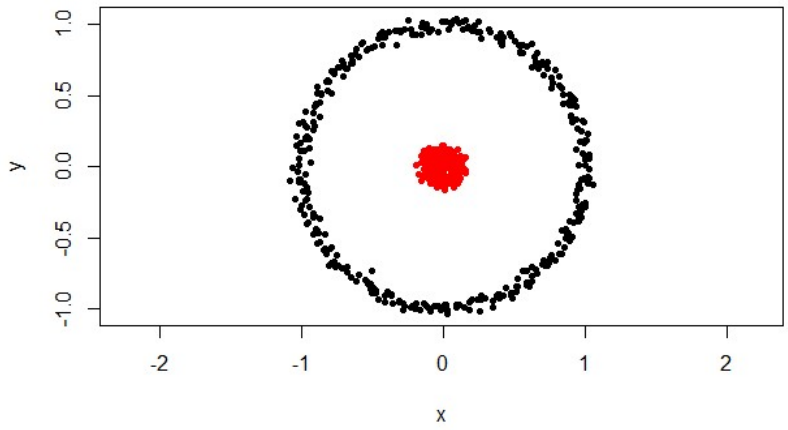
Finally, the implementation of the kernel k-means described in this project is used against the synthetic data. Recall that, to start this method, a random initialization of the clusters is required. In Figure 4, this initial random initialization the matrix of kernels is presented. It is shown that the points are randomly assigned to some of the 2 clusters.

Figure 4: Initialization clustering.



After 21 iterations, the method converges and the clustering presented in Figure 5 is obtained. Note that, to be able to compare our implementation with the previous one, we have used again the RBF kernel function in our method.

Figure 5: Kernel k-means (own implementation).



According to the results, the implementation in R of the kernel k-means algorithm that has been detailed in this document, obtains the same clustering results as the implementation done in 'kernlab'. Table 2 summarizes the performance in terms of execution time and accuracy.

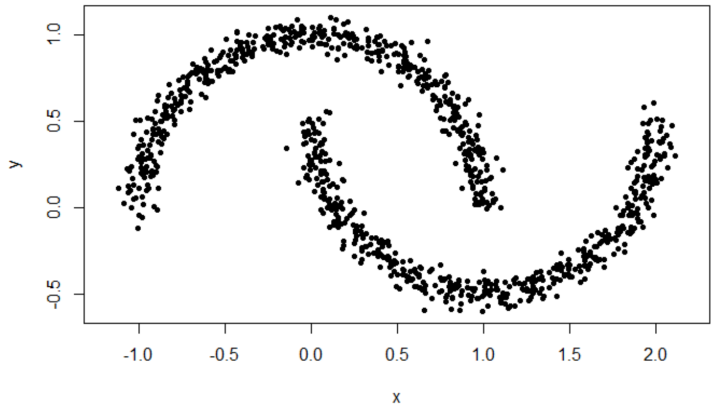
Table 2: Performance on the rings data.

Standard k-means (library 'kernlab')	0.02 secs	70.43% accuracy
RBF Kernel k-means (library 'kernlab')	0.56 secs	100% accuracy
RBF Kernel k-means (own implementation)	25.78 secs	100% accuracy

Moons dataset

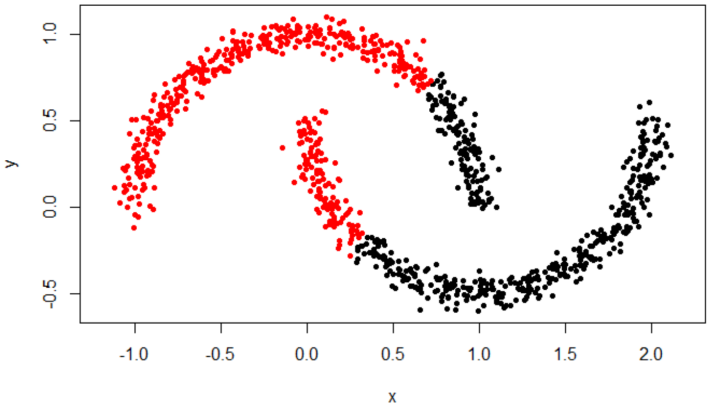
The second dataset used to compare the performance of these methods consists of two half-moons. This 2-D dataset, which is more complex to cluster than the previous one even using a kernel function, is presented in Figure 6.

Figure 6: Moons dataset.



The first method is again the classical k-means algorithm, already implemented in library 'kernlab'. The clustering obtained is presented below:

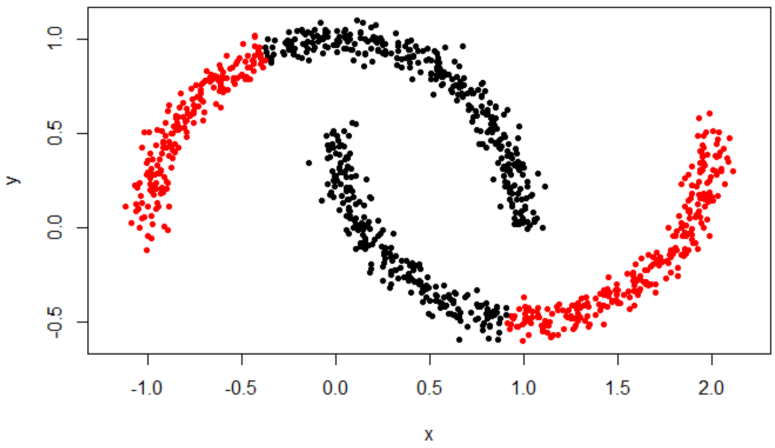
Figure 7: Moons dataset k-means clustering.



As expected, since the clusters are not linearly separable, the classical k-means fails to cluster the two groups of data points.

The next algorithm to be tested is the kernel k-means that is already implemented in the package 'kernlab'. Again, we use the Radial Basis kernel Function (RBF). The clustering obtained is presented in the following figure:

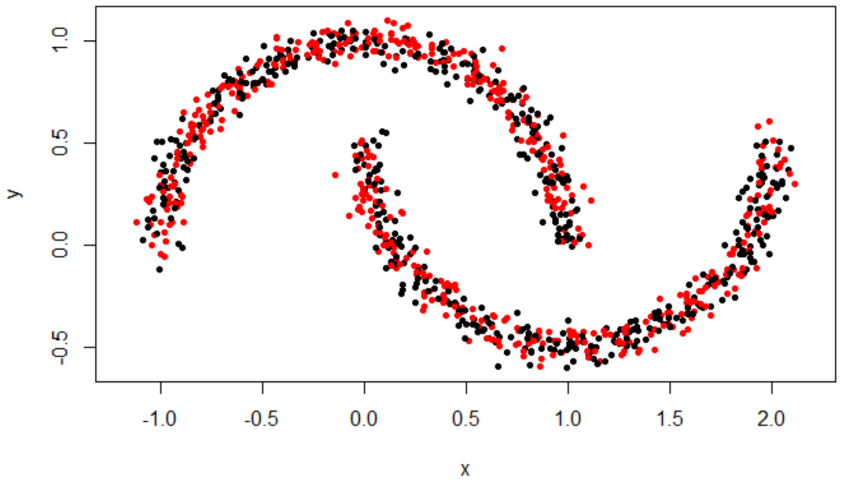
Figure 8: Moons dataset kernel k-means clustering.



We can see that, for this problem, not even the kernelized version is capable to cluster the data points in two correct clusters.

Finally, the implementation of the kernel k-means described in this project is used against the rings data. Recall that, to start this method, a random initialization of the clusters is required.

Figure 9: Initialization clustering.



After 14 iterations, the method converges and the clustering presented in Figure 10 is obtained.

Figure 10: Kernel k-means (own implementation).

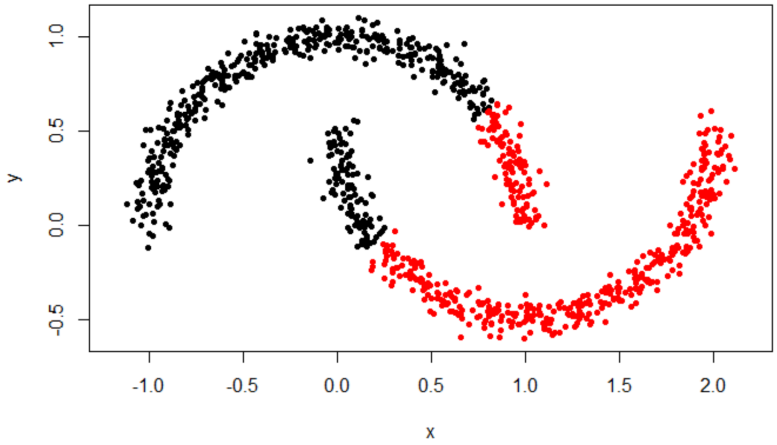


Table 3 summarizes the performance in terms of execution time and accuracy.

Table 3: Performance on the moons data.

Standard k-means (library 'kernlab')	0.02 secs	62.60% accuracy
RBF Kernel k-means (library 'kernlab')	1.03 secs	66.10% accuracy
RBF Kernel k-means (own implementation)	55.86 secs	89.60% accuracy

6 Future work

As explained at the beginning of this study, k-means suffers from different drawbacks. In this paper, we have focused on addressing the difficulties that the standard k-means faces when dealing with non-linearly separable clusters.

However, as a future work, it would be interesting to study a new version of k-means that faces other drawbacks, such as the strong dependency that the final solution has in the initialization of the centroids. In this sense, there is a version of the original k-means, called 'global kernel k-means' [2], that focuses on addressing both the non-linearly separable clusters problem, as well as the initialization dependency.

7 Conclusions

In this paper we have detailed the implementation of a kernelized version of the popular clustering method k-means. The aim of this modification is to obtain an algorithm capable of identifying non-linearly separable clusters.

After the results obtained in the evaluation of this algorithm, we stress the following conclusions:

- When applying our own implementation on the rings dataset, we obtained the same accuracy as using the kernel k-means implemented in 'kernlab'. With both methods we obtain a 100% accuracy, meaning that the algorithm is perfectly capable to identify

the two clusters. This overcomes the standard k-means, which fails to differentiate non-linearly separable clusters. This last method obtained an accuracy of 70.40%.

- When testing the performance on the moons dataset, none of the methods was capable to perfectly separate the two clusters. Note that, because of the 'half-moon' shape, other techniques such as spectral clustering is more suitable [5]. The standard k-means, obtained a poor accuracy equal to 62.60%. The kernel k-means implemented in 'kernlab', does not significantly improve this performance, and obtained an accuracy of 66.10%. Our implementation, even though is not capable to separate properly the two clusters, is the method that obtained the higher accuracy, 89.60%.

- The main difference among the three algorithms compared in this work, is in terms of execution time. In the results obtained for both datasets, the standard k-means is, by far, the fastest algorithm. This is reasonable because of the simplicity of the method. Of course, its main drawback is the poor accuracy obtained for non-linearly separable clusters. On the other side, our implementation is the one obtaining the best accuracy. However, this is at the expense of a much higher execution time. In the middle, the 'kernlab' implementation of kernel k-means runs very fast, but obtains a poor accuracy of 66.10%. Note that if we cluster the data points totally at random, we would get, on average, an accuracy of 50% since half of the points belong to one cluster and the other half, to the other.

- Therefore, we can conclude, that our method is robust, and is capable of obtaining a great accuracy when dealing with complex and non-linearly separable clusters. However, this is obtained at the expense of a higher execution time.

References

[1] B. Scholkopf, A. Smola, and K.-R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation, 10:1299–1319, 1998.

[2] G. F. Tzortzis and A. C. Likas, "The Global Kernel k-Means Algorithm for Clustering in Feature Space," in IEEE Transactions on Neural Networks, vol. 20, no. 7, pp. 1181-1194, July 2009. doi: 10.1109/TNN.2009.2019722

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=5033312isnumber=5159575

[3] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2004. Kernel k-means: spectral clustering and normalized cuts. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '04). ACM, New York, NY, USA, 551-556. DOI=http://dx.doi.org/10.1145/1014052.1014118

[4] N. Cristianini and J. Shawe-Taylor. Introduction to Support Vector Machines: And Other Kernel-Based Learning Methods. Cambridge University Press, Cambridge, U.K., 2000.

[5] Wang, Xiang, Buyue Qian, and Ian Davidson. "On constrained spectral clustering and its applications." Data Mining and Knowledge Discovery 28, no. 1 (2014): 1-30.