Normalized Cut: Theory and Applications

Fahad Ahmed

ID: 610877

Program: Master's in Economics and Management Science

Statistical Software: R

Berlin, Germany, 19.03.2021

This paper is an independent research work on Normalized Cut (NCUT) and by extension, Spectral Clustering. In this work, I lay the theoretical foundations of NCUT and will continue to apply the NCUT algorithm on artificially produced datasets. In the theoretical part of the paper, I briefly describe the idea of Spectral Clustering and discuss graph theory to establish the group for normalized cut algorithm via graph laplacians and continue to explain graph cuts as novel ways to parititon the data. Following basic theory, I draw correspondence between the original normalized cut optimization problem (which is difficult to solve) with a few relaxed version of the problem and how it simplifies the optimization problem. I obviously apply these ideas to an artificial dataset and perform some convergence analysis. s

1 On Spectral Clustering

NCUT was built under the umbrella of Spectral Clustering.In recent years, Spectral Clustering has become one of the most prevalent clustering algorithms in modern data analyses. is a generalization of standard clustering methods. It is imperative to first mention some useful clustering methods, such as K-means or K-medoids – not to excoriate these methods, but rather to point out some advantages that Spectral Clustering has over these traditional methods – that are prevalent in unsupervised machine learning. However, such methods use a spherical or elliptical metric to group data points. Therefore, they do not perform well under nonconvexities. See, for example, the figures 1 and 2 on the next page, where its obvious that typical clustering methods might underperform due to apparent nonconvexities in data.

2 On Graph Theory

Spectral Clustering has its roots in graph theory. I shall first shed some light on graph theory before moving on to the main topic. A graph is defined as G = V, E, where E is the set of edges that connect

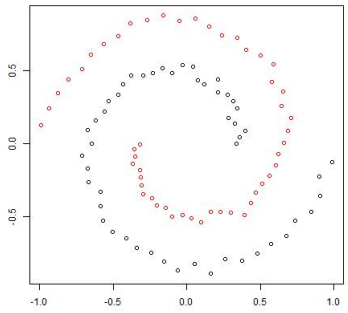

Figure 1: This figure has been produced via the 'mlbench' package in R

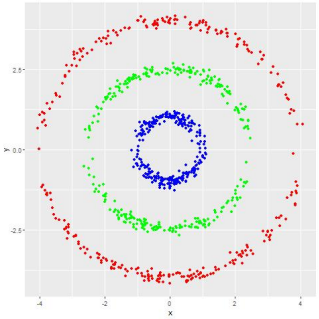

Figure 2: The figure has been simulated manually in R. The code has been provided in the appendix

the nodes V.
A simple graph can be modified to add weights to the edges. A weight is a numerical value assigned to each edge in the graph. For this paper, I assume that G is an undirected, weighted graph. Typically, these values are nonnegative and can represent:

- Cost or distance, for instance, the amount of effort needed to travel from one place (read 'node') to another, or,
- Capacity, that refers to the maximum amount of flow, for instance energy, (or information) that can be transported from one place to another

2.1 Graph Laplacians

Spectral Clustering is performed via Laplacian matrices. There is no one specific conventional 'graph laplacian' matrix, but there are several variants of graph laplacians. The matrix of weights W = (wij) from the weighted graph is called the adjacency matrix that tells us how well a graph is connected – the elements wij take nonzero values if there is a connection between vertices. Also of importance is the degree matrix D, which is a diagonal matrix whose elements represent the degree of each vertex i computed as the sum of weights of all the edges connected to it. I shall now describe two types of graph laplacians:

- Unnormalized graph laplacian:

  The unnormalised graph laplacian can be simply defined as L = D-W, where the matrix of weights W = (wij) from the weighted graph is called the adjacency matrix that tells us how well a graph is connected – the elements wij take nonzero values if there is a connection between vertices. Also of importance is the degree matrix D, which is a diagonal matrix whose elements represent the degree of each vertex i computed as the sum of weights of all the edges connected to it. The L matrix is symmetric and positive sem-definite. The smallest eigenvalue of L is 0 and the corresponding eigenvector is the constant one vector. For any vector $x \in R^n$, we can write the quadratic form $v'Lv$ as:

  $$v'Lv = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}\left(v_i - v_j\right)^2$$

  We can see that the quadratic form will be small if there are large adjacencies between pairs of points and when they are close to each other. In other words, the sum above will vanish only if all terms vanish. If we assume that v is an eigenvector with eigenvalue 0, then the sum above will equal 0. Since the weights are non-negative, the sum above will vanish (be small) if $w_{ij}\left(f_i - f_j\right)^2$ for all $i, j \in 1, 2, ...n$ disappear (be zero). Therefore, the eigvenvector v must be constant for all vertices in order for this sum to equal to 0 if two vertices are connected by a path in the graph. In other words, v should be constant across the vertces in a connectd component.

  Assume that in an undirected, weighted graph G, there are k connected components, we can organize the adjacency matrix W such that it is a block diagonal matrix with each element L_i of that matrix being a proper graph laplacian on its own.

  $$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{pmatrix}.$$

  Since each element in the matrix above is a graph laplacian, it has an eigenvalue 0 with constant 1 as the corresponding eigenvector. Therefore, the multiplicity of 0 eigevenalues of L equals the number of connected components.

- Normalized graph laplacian:

  In literature, there are two prevalent ways one can construct normalized laplacians :

  - $L_sym := D^-1/2LD - 1/2 = I - D^-1/2WD^-1/2$
  - $L_rw := D^-1L = I - D^{-1}W$

  The first matrix is a symmetric matrix, hence the notation 'sym' while the second matrix is closely related to a random walk, hence the notation 'rw'. The matrix of weights (also called 'affinity') W = (wij) from the weighted graph is called the adjacency matrix that tells us how well a graph is connected – the elements wij take nonzero values if there is a connection between vertices. Also of importance is the degree matrix D, which is a diagonal matrix whose elements represent the degree of each vertex i computed as the sum of weights of all the edges connected to it.

For any vector $v \in R^n$, we can write:

$$f' L_{\text{sym}} f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2$$

We can replace L_{sym} with L_{rw} in the formula above. And as in unnormalized laplacian, the multiplicity of the eigenvalue 0 of the normalized laplacian is associated with the number of connected components, that is, the in an undirected, weighted graph G, the multiplicity of the eigenvalue 0 is equal to the number of connected components in the graph.

I shall now move toward the main topic of this paper, which is to contextualize Spectral Clustering in terms of Graph Cuts.

2.2 Graph Cuts

Graph cuts can be viewed as a grouping method based on the idea of graph partitions. Consider a point cloud with N observations. We can represent the observations in an undirected similarity graph G= V, E as N vertices v_i where i belongs to 1,2,. . . ,N. We connect the pairs of vertices with edges if the corresponding observations are similar. We can further assign weights w_{ij} to these edges that represent the degree of similarity between the vertices. We will later get into further details of how similarity between the vertices can be gauged.

With vertices representing data points and the numerical value of edges representing the degree of similarity, we can transform the clustering problem as a graph-partitioning problem. The idea is to divide the graph into sub-graphs such that within-cluster variation is minimised (meaning that points within the same cluster are similart to each other) and between-cluster variation is maximised (meaning that points that do not belong to the same cluster are dissimilar). Consider a weighted graph G = (V, E, W). We employ the graph-cut technique where the degree of dissimilarity between, say, two groups is computed as the total weight of edges removed between them, such that edges within a group have large weights. i.e. vertices are similar, and edges across groups have small weights, i.e. vertices are dissimilar. There are three major types of graphs that deal with locality differently:

- *The ϵ-neighbourhood graph*: In this graph, we connect points whose pairwise distance is within a ball of radius ε. A drawback of this graph is that all points that are connected are given the same weights. Hence, this graph is an ε-neighbourhood is usually considered as an unweighted graph.

- *k-nearest neighbour graph*: We construct this graph by connecting vertex v_i with v_j if the latter is among the k-nearest neighbours of the former. Weights are assigned to the edges by the similarity of their endpoints.
- *The fully connected graph*: As the name suggests, we connect all points in the graph with positive similarity. This graph represents the local neighbourhood relations via a similarity functions.

Let us now treat the aforementioned ideas more concretely. Let X be a set data points X = x1, ..., xn such that x_i belongs to R^p. The similarity matrix S = (s_{ij}), where s_{ij} ¿0 states the similarity between points xi and xj in Spectral Clustering. To construct an appropriate similarity matrix, a kernel function must be chosen. One commonly used function used to construct S, i.e. $s_{ij} = K(x_i, x_j)$, is the Gaussian Kernel $s_{ij} = \exp\left(-\|x_i - x_j\|^2 / \left(2\sigma^2\right)\right)$
Other possible functions are the simple Euclidean distance and the correlation function.

With the aforementioned theoretical setup, we can now move on to define the optimization problem 'mincut'.

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

However, this criterion is not efficient for practice. Wu and Leahy (paper) discovered the mincut criteria occasionally supports paritioning isolated nodes in the graph. This unnatural bias occurs because smaller segments of the graph by definition usually have a small number of connections. This is not what we want to achieve as clusters should be large clouds of data points.
There are several ways we can deal with this issue at hand. One way is to explicitly program that the resulting clusters are large. A common objective function to encode this is the RatioCut propsed by Hagen and Kahng (1992), which measures the size of a subset of a graph by the number of vertices. Another renowned method was proposed by Shi and Malik (2000) to paritiion the graph. They call this new disassociation measure the Normalized Cut (Ncut), where the size of a subgraph is measured by the weights of the connected edges.
Following is the definition of the Ncut problem:

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij} \left(\frac{1}{vol(A)} + \frac{1}{vol(B)}\right)$$

Where vol(A) is the total edge connections from nodes in A to all nodes in graph and vol(B) is the total edge connections from nodes in B to all nodes in the graph. Why does Ncut help to correct the bias in cut's algorithm? The reason is that cutting out small isolated points will not achieve a small Ncut value since the cut value is divided by number of edge connections from that small set to all other nodes. Notice that the minimum of the objective function is achieved if volumes coincide. Essentially, what the function tries to achieve is balanced clusters. However, introducing this complexity for more efficient results complicates the optimization problem. We can employ Spectral Clustering as a way to solve relaxed versions of the Ncut problem.

It has been established that relaxing Ncut leads to normalized spectral clustering as elaborated above. However, there is no gaurantee that the solution of the relaxed version will be very close to the solution of the original ncut problem. The reason why SPectral Clustering is still widely used is not because of the quality of its solutions, but that it yields a standard linear algebra constrained optimization problem, which is rather simple to solve.

2.3 Random Walks and Ncut

We can draw correspondence between random walks and the idea of Ncut.In graph theory, a random walk is a process that randomly jumps from one vertex to another. We can interpret spectral clustering as atttempting to parition the graph such that the random walk on the graph stays within the same cluster for as long as possible and rarely switches between clusters. If we recall the discussion above, this makes sense since a balanced parition with a low Ncut measure should have the property that the random walk does not have many chances to switch clusters. In more formal terms, the transition probability of going from one vertex to another vertex is assigned by $p_{ij} = w_{ij}/d_i$. Therefore, the transition matrix is then given by:

$P = D^{-1}W$.

If the graph is not bipartite and is connected, then the random walk yields a unique stationary distribution. Naturally, there is a strong relation between the transition matrix P and L_{rw} defined above: $L_{rw} = I - P$

By extension, if λ is an eigenvalue of L_{rw} with eigenvector v, then 1-λ is an eigenvalue of P with the same eigenvector.

We can view the Ncut problem via transition probabilities in random walks. In a connected and non-bipartite graph G, assume that we start a random walk with a stationary distribution π. We have that:

$$Ncut(A, B) = P(A|B) + P(B|A), \text{ where B is the complement of A}$$

This gives us a different interpretation of normalized spectral clustering and of Ncut. It is clear that while minimizing Ncut, we search for a graph cut such that a random walk does not transition from A to B or vice versa.

I will not establish formal equivalence of Ncut and transition probabilities since that is out of the scope of this paper. For further details and formal proof, it is advisable to go through Meila and Shi (2001).

2.4 Ncut and Hyperplanes

It can be shown that Ncut shifts the data set to an infinite-dimensional feature space and partitions the data through a 'gap' in the shifted data. This 'gap' can be some intuitive measure of distance between a plane and set of points with an arbitrary feature space. The Ncut algorithm implicitly focuses on the behaviour of data points away from the mean of the data set, which is why the normalized algorithm can break 'wide' clusters and is sensitive to outliers. We can relax the Ncut algorithm and reinterpret it as a separating hyperplane problem. (http://groups.csail.mit.edu/vision/vip/papers/rahimi-ncut.pdf) show that normalized cuts and clustering methods based on separating hyperplanes perform identical operations. This implies that any geometric inution from separating hyperplane programs will be shared by Ncut algorithms, as well.

3 Application

In this part of the paper, I apply the normalized cut algorithm on an artificial dataset.
How to find A^* and $B^*, (P^*, Q^*) = \underset{Y=P+Q}{\operatorname{argmin}} N_{\text{cut}}\ (P, Q)$?

1. Given an arbitrary feature space, set up a weighted graph G = V, E and assign weights to the edges connecting nodes based on a similarity measure
2. Construct Laplacian matrix L from D (the degree matrix) and W (the matrix of weights)
3. Perform spectral decomposition of $L : L = \Gamma\Lambda\Gamma^{\top}$
4. From the ordered sequence of eigenvectors, use the eigenvector γ_{n-1} that corresponds to the second-smallest eigenvalue in Λ to bipartition the graph
5. Decide if the current partition should be sub-divided, and recursively partition the segments

Ncut is prevalently used in image segmentation and, therefore, most of the algoirhtms available in Python, Matlab and R revolve are made for image data. However, I was able to find a readymade ncut algorithm that applies ncut based on graph theory to cluster the columns of an arbitrary dataset. I use the NCutYX package on a 1000*12 dataset in R. For the ncut optimization, it is important that the variables are not categorical or factor variables. The algorithm I employ minimizes the Ncut criterion through the Cross Entropy (CE) method, which is an approximation method.

The ncut functions include several hyperparameters of which of utmost interest are the number of predefined clusters K and the number of iterations. I test the convergence of the algorithm by defining a sequence of iterations and checking at what point the numerical ncut measure stops changing. Naturally, the numerical ncut measure would increase the ncut measure, which is why I test convergence by setting K = 2 and K = 3. As explained above, there are several ways to construct a similarity matrix. The ncut function I use allow us to set the edge weights according to 'correlation', 'euclidean' distance and 'gaussian' kernel.

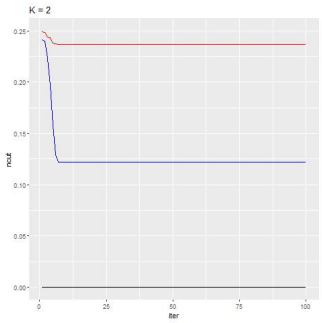

Figure 3: Blue = 'Correlation', Red = 'Euclidean', Black = 'Gaussian'

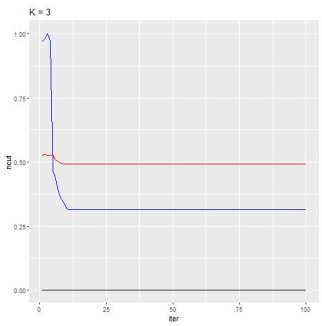

Figure 4: Blue = 'Correlation', Red = 'Euclidean', Black = 'Gaussian'

The figures posted below are used for convergence analysis. First of all, it is natural that the ncut measure increases with the number of clusters. However, changing the number of clusters does not affect the performance of the algorithm significantly. In both cases, the gaussian kernel outperforms and yields a very small ncut value. When I use 'correlation' for similarity, the convergence is quicker in both cases, but it yields larger ncut value as opposed to when the simple 'euclidean' distance is used as kernel. Recall that relaxing the ncut problem leads to normalized spectral clustering. Therefore, I shall now perform spectral clustering algorithms on the same dataset and compare the results wiht traditional clustering algorithms, such as k-means.

First, let us look at figures 5 and 6 below. In both the figures, I cluster the spiral data from mlbench package in R (check above). In figure 5, I use the spectral clustering algorithm from the package 'kernlab' in R. It is obvious that the algorithm paritions the data perfectly when number of clusters is equal to 2. In figure 6, the parition has been performed using the tradional k-means algorithm from the 'kknn' package in R. However, this algorithm performs significyntly worse as compared to spectral clustering. Therefore, this serves as a visual example of how spectral clustering algorithms are superior in presence of nonconvexities.

I performed the clustering analysis for spirals data using unnormalized graph laplacians. The point was merely to show where spectral clustering should be used instead of other clustering methods. I now move on to discuss a relaxed version of the normalized cut problem, which - as stated above - can take form of normalized spectral clustering. To perform normalized spectral clustering, I use (after slightly modifying) code from github available at normalized spectral clustering algorithm. I discussed three types of graph above, namely: e-neighborhood; k-nearest neighbor graph; and the fully connected

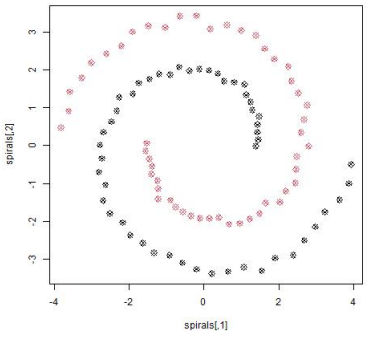

Figure 5: clustering 'spirals' from mlbench using spectral clustering

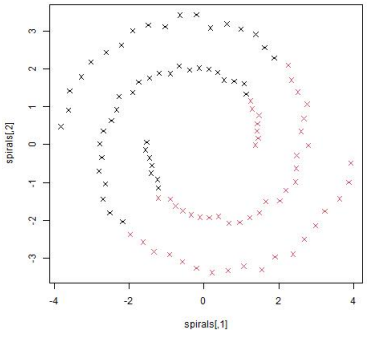

Figure 6: clustering 'spirals' from mlbench using k-means

graph. The algorithm I employ constructs a k-nearest neighbor graph. Furthermore, normalization is performed using the L_{sym} matrix defined above.

I plotted the figures below using the fviz_cluster() function in R from 'factortoextra' package to display some clustering results visually. The figures have been plotted for k =4, k = 8, and k = 12, respectively.

Figures 10 and 11 show the total within sum of squares plotted against the number of clusters. I did not run the clustering algorithm on to different datasets, but rather twice on the same dataset to check how much the results vary. Overall, one can definitively say that increasing the number of clusters improves the performance as evident by the decreasing within sum of squares. However, both graphs are elbow-shaped, meaning that total within sum of squares decrease rapidly until a certain point and then the gradient of the graph significantly reduces as the number of clusters increase. We can say that the optimal number of clusters lies somewhere betweem the range of 5 to 10 based on the graphs.

I use the last two figures, namely, figures to comment on convergence and stability of the algorithm. It is reasonable to assume that the algorithm will take longer to run as the number of clusters increase. However, the last two figures show no apparent signs that this might be the case. A possible reason is that there are several extra steps, such as construction of graph laplacians, computation of eigenvectors and so and so forth, which is why the elapsed time does not only depend on the number of clusters. Therefore, in terms of time, we cannot definitively make a statement regarding the algorithm.

4 Conclusion

I have succesfully built the theoretical background for spectral clustering and have applied several versions of the algorithms on the provded dataset for comparison. A drawback of this paper is that the algorithm was not applied to actual data from the images. Apart from that, I have shed light on major points of the topic.

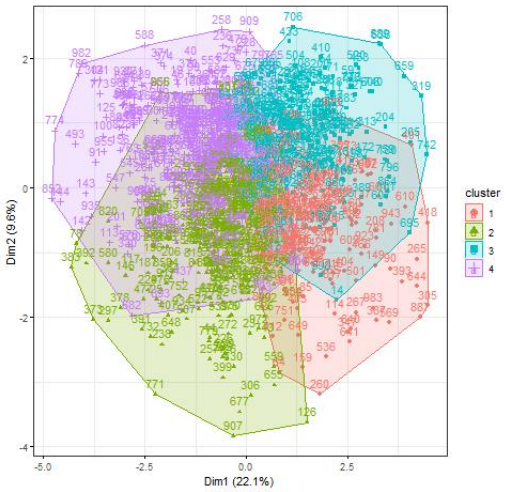

Figure 7: K = 4

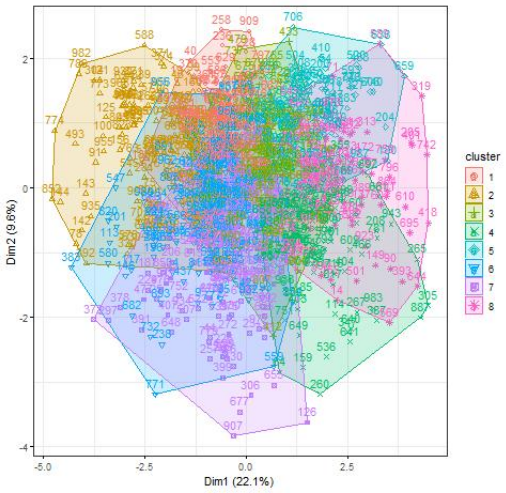

Figure 8: K = 8

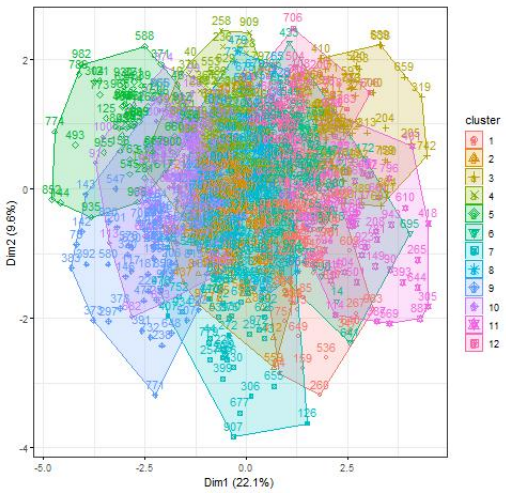

Figure 9: K = 12

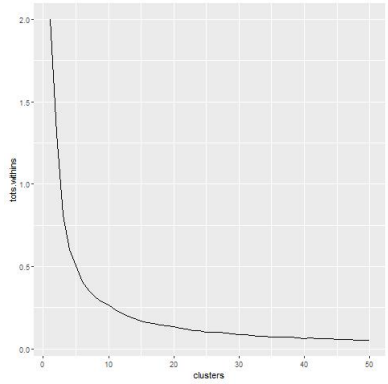

Figure 10:

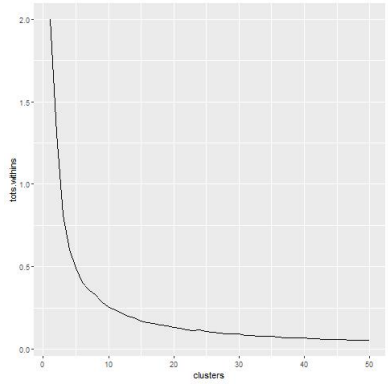

Figure 11:

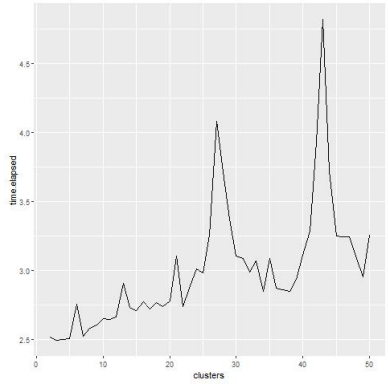

Figure 12: Showing time taken in seconds to complete the clustering algorithm

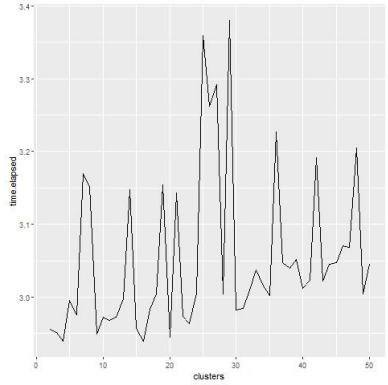

Figure 13: Showing time taken in seconds to complete the clustering algorithm

References

Hagen, L. and Kahng, A., 1992. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9), pp.1074-1085.

Jianbo Shi and Malik, J., 2000. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), pp.888-905.

Luxburg, U., 2007. *A tutorial on spectral clustering | Statistics and Computing*. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1007/s11222-007-9033-z> [Accessed 19 March 2021].

Appendix

```
rm(list = ls())

set.seed(123)

setwd('D:/Academics/Academics/MEMS/Semester 3/MVA/Project')


#loading relevant packages

install.packages(c('kernlab','mlbench', 'NCutYX', 'ggplot2', 'tidyverse', 'factoextra', 'Spectrum'),
dependencies = TRUE)



library(kernlab)

library(kknn)

library(mlbench)

library(NCutYX)

library(ggplot2)

library(tidyverse)

library(factoextra)


##spirals


jpeg('spiral.jpg')

obj <- mlbench.spirals(100, 1, 0.025)

plot(obj)

dev.off()



##simulating concentric cirlces

d <- 2

N <- 300

R <- 4
```

```
x1 <- 1:N %>% map(.f = ~ runif(n = d - 1, min =  - R, max = R))

y1 <- x1 %>% map(.f = ~ c(., sign(runif(n = 1, min = - 1, max = 1))*sqrt(R^2 - norm(., type = '2')^2)))

df1 <- y1 %>% reduce(.f = ~ rbind(.x, .y)) %>%
  as.tibble %>%
  rename(x = V1, y = V2)

sd <- 0.1
y1 <- y1 %>% map(.f =  ~ . + rnorm(n = d, mean = 0, sd = sd))
df1 <- y1 %>% reduce(.f = ~ rbind(.x, .y)) %>%
  as.tibble %>%
  rename(x = V1, y = V2)

ggplot() + geom_point(data=df1, mapping = aes(x = x, y = y))

d <- 2
N <- 300
R <- 1
x2 <- 1:N %>% map(.f = ~ runif(n = d - 1, min =  - R, max = R))

y2 <- x2 %>% map(.f = ~ c(., sign(runif(n = 1, min = - 1, max = 1))*sqrt(R^2 - norm(., type = '2')^2)))

df2 <- y2 %>% reduce(.f = ~ rbind(.x, .y)) %>%
  as.tibble %>%
  rename(x = V1, y = V2)

sd <- 0.1

y2 <- y2 %>% map(.f =  ~ . + rnorm(n = d, mean = 0, sd = sd))
df2 <- y2 %>% reduce(.f = ~ rbind(.x, .y)) %>%
```

```
  as.tibble %>%
  rename(x = V1, y = V2)



ggplot() + geom_point(data=df1, mapping = aes(x = x, y = y),color = 'red') +
  geom_point(data = df2, mapping = aes(x=x, y=y), color = 'blue')

N <- 300
R <- 2.5
x3 <- 1:N %>% map(.f = ~ runif(n = d - 1, min =  - R, max = R))

y3 <- x3 %>% map(.f = ~ c(., sign(runif(n = 1, min = - 1, max = 1))*sqrt(R^2 - norm(., type = '2')^2)))

df3 <- y3 %>% reduce(.f = ~ rbind(.x, .y)) %>%
  as.tibble %>%
  rename(x = V1, y = V2)

sd <- 0.1

y3 <- y3 %>% map(.f =  ~ . + rnorm(n = d, mean = 0, sd = sd))
df3 <- y3 %>% reduce(.f = ~ rbind(.x, .y)) %>%
  as.tibble %>%
  rename(x = V1, y = V2)



jpeg('concentric circles.jpg')

ggplot() + geom_point(data=df1, mapping = aes(x = x, y = y),color = 'red') +
  geom_point(data = df2, mapping = aes(x=x, y=y), color = 'blue')+
  geom_point(data=df3, mapping = aes(x=x, y=y), color = 'green')
dev.off()
```

```
#loading data
data <- read.csv('FahadAhmed.csv', header = TRUE)
data <- scale(data, center = TRUE, scale = TRUE)
data <- data.frame(data)
summary(data)
head(data)



#ncut algorithm

## For K = 2 (2 clusters)



##using correlation as kernel
ncut <- NCutYX::ncut(data,
          K=2,
          B= 100,
          dist='correlation',
          N = 500,
          scale=FALSE,
          q=0.2,
          sigma=1)
corr_2 = data.frame('K' = 2, 'dist' = 'correlation', 'iter' = 1:100, 'ncut' = ncut$quantile)

##using euclidean distance as kernel
ncut <- NCutYX::ncut(data,
                K=2,
```

```
                B= 100,
                dist='euclidean',
                N = 500,
                scale=FALSE,
                q=0.2,
                sigma=1)
euc_2 <- data.frame('K' = 3, 'dist' = 'euclidean', 'iter' = 1:100, 'ncut' = ncut$quantile)

##using gaussan function as kernel
ncut <- NCutYX::ncut(data,
                K=2,
                B= 100,
                dist='gaussian',
                N = 500,
                scale=FALSE,
                q=0.2,
                sigma=1)
gauss_2 <-  data.frame('K' = 3, 'dist' = 'gaussian', 'iter' = 1:100, 'ncut' = ncut$quantile)

jpeg('p.jpg')
ggplot() +
  geom_line(data = corr_2, aes(x = iter, y = ncut), color = 'blue') +
  geom_line(data = euc_2, aes(x = iter, y = ncut), color = 'red') +
  geom_line(data= gauss_2, aes(x=iter, y=ncut), color = 'black') + ggtitle('K = 2')
dev.off()


# K= 3 (3 clusters)
```

```
#the algorithm should be run multiple times in case if it does not work for the first time due to convergence issues in approximation



##using correlation as kernel


ncut <- NCutYX::ncut(data,
                     K=3,
                     B= 100,
                     dist='correlation',
                     N = 500,
                     scale=FALSE,
                     q=0.2,
                     sigma=1)
corr_3 = data.frame('K' = 3, 'dist' = 'correlation', 'iter' = 1:100, 'ncut' = ncut$quantile)



##using euclidean distance as kernel


ncut <- NCutYX::ncut(data,
                     K=3,
                     B= 100,
                     dist='euclidean',
                     N = 500,
                     scale=FALSE,
                     q=0.2,
                     sigma=1)
euc_3 <- data.frame('K' = 3, 'dist' = 'euclidean', 'iter' = 1:100, 'ncut' = ncut$quantile)



##using gaussan function as kernel
```

```
ncut <- NCutYX::ncut(data,
                     K=3,
                     B= 100,
                     dist='gaussian',
                     N = 500,
                     scale=FALSE,
                     q=0.2,
                     sigma=1)
gauss_3 <-  data.frame('K' = 3, 'dist' = 'gaussian', 'iter' = 1:100, 'ncut' = ncut$quantile)



jpeg('s.jpg')
ggplot() +
  geom_line(data = corr_3, aes(x = iter, y = ncut), color = 'blue') +
  geom_line(data = euc_3, aes(x = iter, y = ncut), color = 'red') +
  geom_line(data= gauss_3, aes(x=iter, y=ncut), color = 'black')   + ggtitle('K = 3')
dev.off()



#relaxing Ncut algorithm to perform normalised spectral clustering


##first, some clustering analysis to show the advantage of spectral clustering algorithms over other
algorithms


jpeg('spec.jpg')
spec <- specc(spirals, centers = 2)
plot(spirals, col = spec, pch = 4)
points(spirals, col= obj$classes, pch = 5)
dev.off()
```

```
jpeg('kmeans.jpg')
spec <- kmeans(spirals, centers = 2)
plot(spirals, col = spec$cluster, pch = 4)
dev.off()

##now normalised version

sc <- function(X, # matrix of data points
                        nn = 10, # the k nearest neighbors to consider
                        n_eig = 2) # m number of eignenvectors to keep
{
  mutual_knn_graph <- function(X, nn = 10)
  {
    D <- as.matrix( dist(X) ) # matrix of euclidean distances between data points in X

    # intialize the knn matrix
    knn_mat <- matrix(0,
                      nrow = nrow(X),
                      ncol = nrow(X))

    # find the 10 nearest neighbors for each point
    for (i in 1: nrow(X)) {
      neighbor_index <- order(D[i,])[2:(nn + 1)]
      knn_mat[i,][neighbor_index] <- 1
    }

    # Now we note that i,j are neighbors iff K[i,j] = 1 or K[j,i] = 1
    knn_mat <- knn_mat + t(knn_mat) # find mutual knn

    knn_mat[ knn_mat == 2 ] = 1
```

```
  return(knn_mat)
 }

 graph_laplacian <- function(W, normalized = TRUE)
 {
  stopifnot(nrow(W) == ncol(W))

  g = colSums(W) # degrees of vertices
  n = nrow(W)

  if(normalized)
  {
   D_half = diag(1 / sqrt(g) )
   return( diag(n) - D_half %*% W %*% D_half )
  }
  else
  {
   return( diag(g) - W )
  }
 }

 W = mutual_knn_graph(X) # 1. matrix of similarities
 L = graph_laplacian(W) # 2. compute graph laplacian
 ei = eigen(L, symmetric = TRUE) # 3. Compute the eigenvectors and values of L
 n = nrow(L)
 return(ei$vectors[,(n - n_eig):(n - 1)]) # return the eigenvectors of the n_eig smallest eigenvalues

}

#list of clustering solutions
clusters <- list()
```

```
#collects data for total within sum of squares
withins <- data.frame('tots.withins' = NA, 'clusters' = 1:50)

#collects data for aglorithm run time
time <- data.frame('time.elapsed' = 0, 'clusters' = 1:50)
for(i in 1:nrow(time)) {

  start_time <- Sys.time()
  sc <- spectral_clustering(data)
  kmeans <- kmeans(sc, time$clusters[i])
  end_time <- Sys.time()
  time_taken <- end_time - start_time
  time$time.elapsed[i] <- time_taken
  clusters[[i]] <- kmeans
  withins$tots.withins[i] = kmeans$tot.withinss
}
time <- time[2:50,]


#saving all plots into jpeg format
jpeg('time.jpg')
ggplot(time) + geom_line(aes(x=clusters, y = time.elapsed))
dev.off()

jpeg('time1.jpg')
ggplot(time) + geom_line(aes(x=clusters, y = time.elapsed))
dev.off()


jpeg('4.jpg')
```

```
fviz_cluster(clusters[[4]], data= data, ggtheme = theme_bw(), main = "")
dev.off()

jpeg('8.jpg')
fviz_cluster(clusters[[8]], data= data, ggtheme = theme_bw(), main = "")
dev.off()

jpeg('12.jpg')
fviz_cluster(clusters[[12]], data= data, ggtheme = theme_bw(), main = "")
dev.off()

jpeg('withins.jpg')
ggplot(withins) + geom_line(aes(x=clusters, y=tots.withins))
dev.off()

jpeg('withins1.jpg')
ggplot(withins) + geom_line(aes(x=clusters, y=tots.withins))
dev.off()-
```

Declaration

I hereby confirm that I have authored this document independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such.

I have attached the code used to produce the analysis in the appendix. I confirm that I have written and executed the analysis, and that the code is complete and executable

BERLIN, GERMANY, 19.03.2021

Fahad Ahmed