University Grenoble Alpes

Modelling Seminars and Projects

Electricity Load Clustering

Vasilii Feofanov
Michał Lewandowski
Amir Asarbaev
Mouna Rhalimi

supervised by
Emilie Devijver

January 30, 2018

Contents

Introduction **3**

1 Functional Data Analysis **4**
1.1 Functional Data . . . 4
1.1.1 Random functions . . . 4
1.1.2 Functional data sample . . . 4
1.2 Functional data representation via basis expansion . . . 5
1.2.1 Basis Expansion . . . 5
1.2.2 B-Splines . . . 5
1.2.3 Data smoothing . . . 6
1.3 Functional Principal Component Analysis . . . 6
1.3.1 Decomposition of random functions . . . 6
1.3.2 A case of real data . . . 7

2 Cluster Analysis **8**
2.1 Formulation of the Problem . . . 8
2.2 K-Means . . . 8
2.2.1 The algorithm . . . 8
2.2.2 The k-means plus plus . . . 9
2.2.3 Pros and cons . . . 9
2.3 Hierarchical Clustering . . . 9
2.3.1 The algorithm . . . 9
2.3.2 Type of Linkage . . . 10
2.3.3 Pros and Cons . . . 10
2.4 Expectation-Maximization Algorithm . . . 10
2.4.1 Principle of the EM . . . 10
2.4.2 Mixture of Gaussian models . . . 11
2.4.3 The case of diagonal covariance matrices . . . 11
2.4.4 Pros and cons . . . 12
2.5 Spectral Clustering . . . 12
2.5.1 The algorithm . . . 12
2.5.2 Justification . . . 13
2.5.3 Advantages and disadvantages . . . 13
2.6 Methods for identification the number of clusters . . . 13
2.6.1 Partitioning around medoids . . . 13
2.6.2 Calinsky criterion . . . 14
2.6.3 Affinity propagation . . . 14
2.6.4 Davies–Bouldin index . . . 15
2.6.5 The gap statistic . . . 16
2.6.6 The slope heuristic . . . 16

3 Clustering of Electricity Load Consumption **17**
3.1 Data description 17
3.2 Clustering data 18
3.2.1 Introduction 18
3.2.2 Estimation of the best number of clusters 18
3.3 Clustering algorithm implementation 25
3.3.1 Introduction 25
3.3.2 k-means++ 25
3.3.3 The EM algorithm 27
3.3.4 Hierarchical clustering 28
3.3.5 Spectral clustering 30

Conclusion **31**

Bibliography **32**

Appendix **33**

Introduction

The cluster analysis plays an important role in various applications today. Particularly, it is widely used as one of the step of data exploration. In clustering we aim to find similarities between observations, and, based on this, form homogeneous groups in such a way that an object is closer to observations from the same group than to other ones. This type of analysis may contribute to better understanding of the data structure with a view to interpret it and discover hidden obstacles.

Nowadays, clustering problem is well developed. There exists a lot of methods that propose a solution. It can be connectivity-based algorithm (hierarchical clustering), grouping around centroids (k-means), or a method that estimates distribution of the model (the EM algorithm). However, in real applications different issues are arose in the analysis. One the most common is the curse of dimensionality. A particular case can be data of curves. In this case, the sample data are realizations of a random function that is defined on a continuous set. A set of problems that work with such data is called functional data analysis.

Today, the analysis of functional data is a topical problem. Its application includes economics, meteorology, medicine and other domains [1, 2, 3]. In this paper we consider one specific issue, namely, daily consumption of electricity. Each observation represents summarised by consumers electricity load through a day. Such aggregated load has a property to be smooth, thereby the data can be considered as functional.

The aim of this paper is to study techniques of the cluster analysis in a situation when data is functional. More specifically, we want to find partition of the electricity load data. To cluster this data properly, here, we consider dimension reduction procedure. It can be b-spline projection or functional principal component analysis. This technique leads to the case of multivariate analysis that is well studied for the cluster analysis.

This paper is organized as follows. Chapter 1 introduces functional data and tools that are closely connected to it, namely, b-spline approximation as well as functional principal component analysis. Chapter 2 describes popular clustering algorithms. In addition, methods of model selection and comparison are reviewed. Chapter 3 presents statement of the electricity load clustering problem. Then, the experiment description and numerical results are presented. Finally, results are summarised in conclusion.

Chapter 1

Functional Data Analysis

1.1 Functional Data

1.1.1 Random functions

Before introducing to functional data, let's consider preliminary notation of random functions.

A measurable function X is called a random function when it maps the Cartesian product of a sample space Ω and a set $\mathcal{T}$ to a set $\mathcal{X}$:

$$X : (\Omega, \mathcal{T}) \to \mathcal{X}.$$

Further, we consider $t \in \mathcal{T} = \mathbb{R}^+$ as a time and let $X(\cdot, t) \equiv X(t)$ be a square integrable L^2-function:

$$\left[\int X^2(t)\mathrm{dt}\right]^{1/2} < \infty$$

$$\langle X, Y\rangle = \int X(t)Y(t)\mathrm{dt}; \quad \|X\| = \sqrt{\langle X, X\rangle}$$

Similarly to a random variable, mean, variance as well as covariance can be proposed for random functions. However, in this case, they also will be functions:

$$\mu(t) = \mathbb{E}X(t); \quad \varsigma(t) = \mathbb{E}[X(t) - \mu(t)]^2;$$

$$\sigma(t, s) = \mathbb{E}[X(t) - \mu(t)][X(s) - \mu(s)], \quad t, s \in \mathcal{T}.$$

1.1.2 Functional data sample

Thus, we can define a functional data sample of size n as a set of mutually independent n realizations of a random function X. We denote this sample as $x(t) = \{x_i(t)\}_{i=1}^n$. These observations, which are actually curves, will be the primary object of analysis. Now, we can introduce descriptive statistics of the sample data:

$$\bar{x}(t) = \frac{1}{n}\sum_{i=1}^{n} x_i(t); \quad \varsigma(t) = \frac{1}{n-1}\sum_{i=1}^{n}[x_i(t) - \bar{x}(t)]^2$$

$$\sigma(s, t) = \frac{1}{n-1}\sum_{i=1}^{n}[x_i(s) - \bar{x}(s)][x_i(t) - \bar{x}(t)]$$

In fact, in real applications we usually are not able to have the genuine shape of a curve. As a rule, we have access only to some values of an observation that were measured with a period of time. Therefore, instead of curves we commonly have measurements of these curves that were derived in a time span $[a, b]$ with some period l:

$$t_1 = a, \ t_2 = a + l, \ \ldots, \ t_{p-1} = a + (p-2)l, \ t_p = a + (p-1)l = b.$$

$$\mathbf{x} = \{x_{ij}\}, \quad i = \overline{1,n},\ j = \overline{1,p}.$$

Now, there is a question how to analyse this discretized data. There are two approaches. The first one is to work with the data as in the multivariate analysis. This is a very naive method since each moment of time will be considered as an independent variable. Consequently, we prone to lose common tendencies of curves and rather just make a comparison of values at each considered time moment. Thereby, the second approach suggests to recover the shape of a curve. In this case, we make an approximation of the curve's form given measurements. Thus, we come back to a representation where the sample data is a collection of curves [3].

1.2 Functional data representation via basis expansion

In this section, we discuss details of the curve's shape approximation. A popular and effective way to do it is to smooth measurements by a combination of basis functions that expand an original curve. We introduce one of such methods, namely B-splines.

1.2.1 Basis Expansion

Let's consider a basis of functions such that each curve from the set $x(t) = \{x_i(t)\}_{i=1}^n$ can be represented by a linear combination of theirs. Let $f(t)$ be a vector of these basis functions:

$$f(t) = (f_1(t), \ldots, f_d(t))^T.$$

We consider expansions of curves from $x(t)$ in the matrix form. Then, designating a matrix of coefficients as $\mathbf{C}$, we obtain that $x_i(t)$ and $x(t)$ can be represented as follows:

$$x_i(t) = \sum_{j=1}^{d} c_{ij} f_j(t), \ \ i = \overline{1,n}$$

$$x(t) = \mathbf{C} f(t)$$

$$\mathbf{C} = \begin{pmatrix} \ddots & & \\ & c_{ij} & \\ & & \ddots \end{pmatrix}_{\substack{i=\overline{1,n} \\ j=\overline{1,d}}}$$

Based on this, we can rewrite the sample mean, variance and covariance functions in the following way:

$$\bar{x}(t) = \frac{1}{n}\, \mathbb{1}_n^T \mathbf{C} f(t); \quad \varsigma(t) = \frac{1}{n-1}\, f(t)^T \mathbf{C}^T \mathbf{C}\, f(t)$$

$$\sigma(s,t) = \frac{1}{n-1}\, f(s)^T \mathbf{C}^T \mathbf{C}\, f(t),$$

where $\mathbb{1}_n$ designates a n-size vector, all elements of which are equal to 1.

1.2.2 B-Splines

Let the interval $[a, b]$ of an interest is divided by a sequence of knots:

$$a = \mathfrak{t}_0 < \mathfrak{t}_1 < \cdots < \mathfrak{t}_m < \mathfrak{t}_{m+1} = b.$$

B-splines are basis functions of polynomial form that are connected smoothly in the knots. A result is a smooth piece-wise polynomial in the interval (a, b). Each B-spline has an order h. We define the order as the degree of a polynomial plus one. Thus, basis functions are B-splines that has the order no greater than h and a number of which are $d = m + h$:

$$f_i(t) = B_{i,h}(t), \ \ i = \overline{1, m+h}$$

For construction of B-splines, we augment the knot sequence as follows:

$$\tau_1 = \tau_2 = \cdots = \tau_h = \mathfrak{t}_0$$

$$\tau_{i+h} = \mathfrak{t}_i, \;\; i = \overline{1, m}$$

$$\tau_{h+m+1} = \cdots = \tau_{h+2m} = \mathfrak{t}_{m+1}$$

The B-splines are defined by the Cox-de Boor recursion formula:

$$B_{i,1}(t) = \begin{cases} 1, & \text{if } \; t \in [\tau_i, \tau_{i+1}] \\ 0, & \text{otherwise} \end{cases}, \quad i = \overline{1, m + 2h - 1};$$

$$B_{i,j}(t) = \frac{t - \tau_i}{\tau_{i+j-1} - \tau_i} B_{i,j-1}(t) + \frac{\tau_{i+j} - t}{\tau_{i+j} - \tau_{i+1}} B_{i+1,j-1}(t), \quad i = \overline{1, m + 2h - j}.$$

B-splines expansion is a widely used method that is more appropriate for non-periodical curves [1]. As advantages one can mention its flexibility, smoothness of the curve and computational speed likewise methods with orthogonal basis systems [4].

1.2.3 Data smoothing

Now, we would like to find a way how to estimate the matrix $\mathbf{C}$. One the most popular and simple way to do it is to compute the least square estimation. Let's denote as F a matrix of basis functions' values in moments $t_1, t_2, \ldots, t_p$:

$$\mathbf{F} = \begin{pmatrix} \ddots & & \\ & f_i(t_j) & \\ & & \ddots \end{pmatrix}_{\substack{i=\overline{1,d} \\ j=\overline{1,p}}}.$$

Hence, we reformulate the task as multiple output linear regression:

$$\mathbf{x} = \mathbf{CF}.$$

Then the least square solution is found in the following way:

$$\hat{\mathbf{C}} = \mathbf{x}(\mathbf{F}^T\mathbf{F})^{-1}\mathbf{F}^T.$$

1.3 Functional Principal Component Analysis

1.3.1 Decomposition of random functions

Principal component analysis is a powerful unsupervised approach that is used for exploration of variability, dimension reduction and visualisation. A main idea is to find in the feature space directions along which the variation of the data is explained most. We would like to find a transformation of the feature space such that in the new space first component will explain the variability more than others, the second one more than others except the first one etc. For sake of simplicity, the data is considered with zero mean. In application, if the assumption does not take place, it can be centralised by subtracting the sample mean:

$$X := X - \mu$$

Consider a random function X and its Karhunen-Loève expansion defined by orthogonal functions $\xi_r, \;\; r = \overline{1, \infty}$:

$$X = \sum_{r=1}^{\infty} z_r \xi_r$$

$$\langle \xi_r, \xi_l \rangle = 0, \;\; r \neq l,$$

where $z_r = \langle X, \xi_r \rangle$. We would like to find a basis that is orthonormal and the covariance function of X is explained more when the index r is less. The algorithm of finding this basis can be described as follows [5]:

Algorithm 1 Functional Principal Component Analysis

for $r = 1 : \infty$ **do**

Find ξ_r^* such that:

$$\xi_r^* = \arg\max_{\xi_r} \int \int \xi_r(s)\sigma(s,t)\xi_r(t)ds\,dt$$

A solution is restricted by:

$$\begin{cases} \|\xi_r\|^2 = 1 \\ \langle \xi_r, \xi_l \rangle = 0, \;\; 1 \leq l < r \end{cases}$$

end for

To find ξ_r at each step, it is needed to solve the Fredholm functional eigenequation [4]:

$$\int \sigma(s,t)\xi_r(t)\mathrm{dt} = \lambda_r \xi_r(s),$$

where λ_r is the corresponding eigenvalue.

1.3.2 A case of real data

Suppose that an eigenfunction ξ can be expanded by the basis f with weights $b = (b_1, \ldots, b_d)$:

$$\xi(t) = f(t)^T b$$

Consider a matrix $W = \int f(t)f(t)^T \mathrm{dt}$. Then we can deduce:

$$\int \sigma(s,t)f(t)\mathrm{dt} = \int \frac{1}{n} f(s)^T \mathbf{C}^T \mathbf{C} f(t) f(t)^T b \,\mathrm{dt} = \frac{1}{n} f(s)^T \mathbf{C}^T \mathbf{C} W b.$$

Hence, the eigenequation can be represented as:

$$\frac{1}{n} \mathbf{C}^T \mathbf{C} W b = \lambda b.$$

A condition $\|\xi\|^2 = 1$ is re-written as $b^T W b = 1$. Taking this into account as well as introducing $u = W^{1/2} b$, we get a form:

$$\frac{1}{n} W^{1/2} \mathbf{C}^T \mathbf{C} W^{1/2} u = \lambda u.$$

Thus, u is an eigenvector of the equation's left side. Finding u we can find b by computing $W^{-1/2} u$ [4].

Chapter 2

Cluster Analysis

2.1 Formulation of the Problem

Let X be a sample data that consists of observations x_i with size p: $x_i = (x_i^{(1)}, \ldots, x_i^{(p)})$. The goal is to assign each observation to one of clusters $1, \ldots, K$.

2.2 K-Means

The k-means method is a widely used clustering technique that seeks to minimize the average squared distance between points in the same cluster. Although it offers no accuracy guarantees, its simplicity and speed are very appealing in practice. By augmenting k-means with a very simple, randomized seeding technique, we obtain an algorithm that is $\theta(\log k)$ -competitive with the optimal clustering. Preliminary experiments show that our augmentation improves both the speed and the accuracy of k-means, often quite dramatically.

2.2.1 The algorithm

The k-means method is a simple and fast algorithm that attempts to locally improve an arbitrary k-means clustering. It works as follows. A formal definition of the algorithm is the following:

1. Arbitrarily choose k initial centers $C = \{c_1, c_2, ..., c_k\}$
2. For each $i \in \{1, ..., k\}$, set the cluster C_i to be the set of points in X that are closer to c_i than to any c_j with $j \neq i$.
3. For each $1 \leq i \leq k$, set $c_i = \frac{1}{|Ci|} \sum_{x \in C_i} x$, i.e., the center of mass of the pointsin Ci.
4. Repeat Steps 2 and 3 until the clusters C_i and the centers c_i do not change anymore. The partition of X is the set of clusters $C_1, C_2, ..., C_k$.

It is standard practice to choose the initial centers uniformly at random from X . For Step 2, ties may be broken arbitrarily, as long as the method is consistent.

Note that the algorithm might encounter two possible "degenerate" situations: the first one is when no points are assigned to a center, and in this case that center is removed and we will obtain a partition with fewer than k clusters. The other degeneracy is when a point is equally close to more than one center, and in this case the tie is broken arbitrarily.

In fact, the initialization of centers is an open problem. The performance of the k-means may differ dramatically depending on initial centers. One of the way to make more stable is to initialize k-means by choosing random starting centers with very specific probabilities. Specifically, we choose a point p as a center with probability proportional to p's contribution to the overall potential. This sampling is both fast and simple, and it already achieves approximation guarantees that k-means cannot. It has been proposed to use this approach for seeding the initial centers. It leads to a combined algorithm that is called k-means++.

2.2.2 The k-means plus plus

The k-means++ proposes a specific way of choosing these centers. At any given time, let $D(x)$ denote the shortest distance from a data point x to the closest center we have already chosen. Then, we define the following algorithm, which we call k-means++.

1. Choose an initial center c_1 uniformly at random from X.
2. Choose the next center c_i, selecting $c_i = x' \in X$ with probability $\frac{D(x')^2}{\sum_{x \in X} D(x)^2}$
3. Repeat Step 1b until we have chosen a total of k centers.
4. Proceed as with the standard k-means algorithm (steps 2 - 4 from the k-means method).

We call the weighting used in Step 2 simply "D^2 weighting".

2.2.3 Pros and cons

One can check that the total error ϕ is monotonically decreasing, which ensures that no clustering is repeated during the course of the algorithm. Since there are at most k^n possible clusterings, the process will always terminate. In practice, very few iterations are usually required, which makes the algorithm much faster than most of its competitors.

Unfortunately, the empirical speed and simplicity of the k-means algorithm come at the price of accuracy. There are many natural examples for which the algorithm generates arbitrarily bad clusterings . Furthermore, these examples do not rely on an adversarial placement of the starting centers, and the ratio can be unbounded with high probability even with the standard randomized seeding technique.

The k-means algorithm is attractive in practice because it is simple and it is generally fast. Unfortunately, it is guaranteed only to find a local optimum, which can often be quite poor.

2.3 Hierarchical Clustering

Clustering is dividing a group of items into clusters, while making sure the distance between the items in the same cluster is small. Unlike the k-means clustering, Hierarchical clustering doesn't need to define a certain number of clusters, instead, it generates a tree of different levels of clusters that can be defined as a taxonomy. Hierarchical clustering can be Divisive or Agglomerative. The first type considers the population as one cluster at first, then splits the cluster into multiple ones for each iteration, while the later starts with the observations as single clusters, then merges the clusters with the closest ones in each iteration. In this work, we will focus on the Agglomerative clustering.

2.3.1 The algorithm

The algorithm starts by considering all observations as clusters, then calculate the distance between all these clusters, pick the two clusters that have the minimal distance and merge them into one cluster

1. Turn each cluster into a singleton.
2. For the clusters c1,c2, compute the distance d(c1,c2).
3. merge the pair of clusters that make the smallest distance (depending on the linkage method chosen)
4. Repeat step 2 and 3 until there is one cluster

2.3.2 Type of Linkage

An important step in Hierarchical clustering is computing the distance between clusters. Several methods called linkage are used to compute this distance.

- Single Linkage.
  The proximity between two clusters is the smallest distance between two points, x and y that are in a different clusters called A and B

$$d(A, B) = \min_{x \in A, y \in B} d(x, y) \tag{2.1}$$

- Complete linkage.
  The proximity between two clusters is the biggest distance between two points, x and y that are in a different clusters called A and B

$$d(A, B) = \max_{x \in A, y \in B} d(x, y) \tag{2.2}$$

- Average linkage.

  The proximity between two clusters is the average distance between two points, x and y that are in a different clusters called A and B, while n_j is the number of points in a cluster j

$$d(A, B) = \sum_{x \in A, y \in B} \frac{d(x, y)}{n_A n_B} \tag{2.3}$$

2.3.3 Pros and Cons

In the hierarchical clustering, we don't need to define a certain number of clusters, instead we get all possible levels of clustering using a dendogram. It can show a certain taxonomie. However this type of clustering can be very slow since we don't have an objective function. We can parallelize in order to make it faster.

2.4 Expectation-Maximization Algorithm

An Expectation–Maximization (EM) algorithm is a method that is iteratively looking for a maximum likelihood estimates of model's parameters in a case when the model depends on unobserved latent variables. Parameter estimation for a mixture of normally distributed models is the most known application of this approach. This case is called the Gaussian EM algorithm, which will be described below.

2.4.1 Principle of the EM

From the probabilistic point of view the clustering problem can be formulated as the density estimation task of a mixture model. It means that the probability of outcome x can be represented as:

$$P(x|\theta) = \sum_{j=1}^{K} P(x|z = j, \theta) P(z = j|\theta),$$

where z is a hidden variable that designates to which component of the mixture x belongs to, and θ is a parameter set that is needed to find. We aim to find a solution $\hat{\theta}$ that maximizes the log-likelihood of the sample:

$$\mathcal{L}(\theta, z) = \sum_{i=1}^{n} \log P(x_i|\theta) = \sum_{i=1}^{n} \log \sum_{j=1}^{K} P(x_i|z = j, \theta) P(z = j|\theta)$$

The expectation–maximization algorithm is an approach that helps to find the solution. Starting from the initial value $\theta^{(0)}$ it performs iteratively following actions:

- Compute expected values of posteriors given $\theta^{(0)}$: $P(z = j|x_i, \theta^0)$
- Based on the new posteriors, compute a new estimation of $\theta^{(1)}$ maximizing the log-likelihood.

These two steps are repeated until the moment when log-likelihood will differ from the previous step insignificantly.

2.4.2 Mixture of Gaussian models

Suppose that components (clusters) of a mixture are distributed normally:

$$(x|z = j) \sim \mathcal{N}(\mu_j, \Sigma_j).$$

Then, in this case explicit formulae for the EM algorithm can be inferred. Let's designate prior probabilities as follows: $\pi_j = P(z = j|\theta)$. The Gaussian EM is described in the algorithm 2.

Algorithm 2 Gaussian EM for K clusters

Initialization step: $\hat{\theta} = (\{\hat{\mu}_j\}_{j=1}^K, \{\hat{\Sigma}_j\}_{j=1}^K, \{\hat{\pi}_j\}_{j=1}^{K-1})$, $t = 0$, $\mathcal{L}^{(t)}$, T
repeat
$t = t + 1$
Expectation Step: Compute posterior probabilities

$$\gamma_{ij} = P\{z = j|x_i, \hat{\theta}\} = \frac{\hat{\pi}_j P(x_i|z = j, \hat{\theta})}{\sum_{k=1}^K \hat{\pi}_k P(x_i|z = k, \hat{\theta})}, \;\; i = \overline{1, n}, \; j = \overline{1, K},$$

where $P(x_i|z = j, \hat{\theta})$ is defined as

$$P(x_i|z = j, \hat{\theta}) = \frac{1}{(2\pi)^{p/2}|\hat{\Sigma}_j|^{1/2}} e^{-\frac{1}{2}(x_i - \hat{\mu}_j)^T \hat{\Sigma}_j^{-1}(x_i - \hat{\mu}_j)}$$

Maximization Step: Update estimates of parameters

$$\hat{\mu}_j = \frac{1}{\sum_{i=1}^n \hat{\gamma}_{ij}} \sum_{i=1}^n \hat{\gamma}_{ij} x_i, \;\; j = \overline{1, K}$$

$$\hat{\Sigma}_j = \frac{1}{\sum_{i=1}^n \hat{\gamma}_{ij}} \sum_{i=1}^n \hat{\gamma}_{ij}(x_i - \hat{\mu}_j)(x_i - \hat{\mu}_j)^T, \;\; j = \overline{1, K}$$

$$\hat{\pi}_j = \frac{\sum_{i=1}^n \hat{\gamma}_{ij}}{n}, \;\; j = \overline{1, K-1}$$

$$\hat{\pi}_K = 1 - \sum_{k=1}^{K-1} \hat{\pi}_k$$

until $|\mathcal{L}^{(t)} - \mathcal{L}^{(t-1)}| < \varepsilon$ or $T \leq t$

2.4.3 The case of diagonal covariance matrices

If to assume that all features are independent on each other, we can deduce that covariance matrices of mixture components are diagonal. This fact simplifies computations of the algorithm.

Especially, it concerns the problem of finding the inverse matrix. In the diagonal case it is true that:

$$\Sigma^{-1} = \begin{pmatrix} \sigma^2_{(1)} & & \\ & \dots & \\ & & \sigma^2_{(p)} \end{pmatrix}^{-1} = \begin{pmatrix} \frac{1}{\sigma^2_{(1)}} & & \\ & \dots & \\ & & \frac{1}{\sigma^2_{(p)}} \end{pmatrix},$$

where $\{\sigma^2_{(l)}\}_{l=1}^{p}$ are variances of features $\overline{1,p}$ respectively. Hence, we can store a covariance matrix as a vector $\sigma^2 = \left(\sigma^2_{(l)}\right)_{l=1}^{p}$.

2.4.4 Pros and cons

Since the EM considers the clustering task as determination of a mixture model, it can be inferred in a new form based on the data nature. Also, because of the fact that the algorithm analyses clusters' densities, it is considered as a more powerful method than distance-based clustering approaches such as k-means or hierarchical clustering.

The main disadvantage of this algorithm is that it needs to estimate a big number of parameters, namely $N = K \cdot (p + p(p+1)/2 + 1) - 1$. It means that it becomes impossible to use this algorithm when $N \geq n$. Moreover, even when N is not much smaller that n, it is not correct to use. The reason is a fact that there is a risk of the ill-conditioned covariance matrix for one or more clusters. It leads to a large error for computation of the inverse matrix. As the alternative is to use the diagonal EM that reduces the number of parameters. However, it releases from dependencies between features, which can be an important assumption.

Another drawback is that the optimal solution is not guaranteed. Depending on the choice of initial values, we might easily converge at M-step to the local maximum. To prevent this problem, it is reasonable to run the algorithm several times and choose the value that corresponds to the maximal likelihood.*Need to add references to [6, 7].*

2.5 Spectral Clustering

2.5.1 The algorithm

In the section 2.5.1 the spectral clustering algorithm is presented, then in section 2.5.2 a short heuristic explanation of the chosen method is given. Particular attention was given to so called *spectral clustering* as in the manner presented in [8].

1. Form the affinity matrix $A \in \mathbb{R}^{n \times n}$ defined by $A_{ij} = \exp(-||s_{ij} - s_{ji}||^2/2\sigma^2)$ if $i \neq j$ and $A_{ii} = 0$. Here s_{ij} denotes the slot which is placed on i-th row and j-th column in the matrix of non transformed data.

2. Define D to be diagonal matrix whose (i,i)-element is the sum of A's i-th row and construct the matrix $L = D^{-1/2}AD^{-1/2}$.

3. Find $x_1, \dots, x_k$, the k largest eigenvectors of L (chosen to be orthonormal to each other in the case of repeated eigenvalues), and form the matrix $X = [x_1, \dots, x_k] \in \mathbb{R}^{n \times k}$ by stacking the eigenvectors in the columns.

4. Form the matrix Y from X by renormalizing each of X's rows to have unit length (i.e. $Y_{ij} = X_{ij}/(\sum_j X_{ij}^2)^{1/2}$).

5. Treating each row of Y as a point in $\mathbb{R}^k$, cluster them into k clusters via k-means or any other algorithm (that attempts to minimize distortion).

6. Assign the original point s_i to cluster j if and only if row i of Y was assigned to cluster j.

The scaling parameter σ^2 controls how rapidly the affinity A_{ij} falls off with the distance between s_i and s_j. It was set to $\sigma^2 = 1$, however different values may be taken under consideration. To the authors best knowledge, there does not exist a method which allows to choose automatically the proper value of the parameter.

2.5.2 Justification

An intuitive explanation of spectral clustering will be given, for more technical and detailed motivations reader is refered to [8].

Given the dataset one may want to compute similarity measure between each pair of points. A graph $G = (V, E)$ was constructed, where vertices (V) are the primary data points and edges (E) encodes the information about the similiarity between two given data points. There are many different similarity measures, for this reports purposes Gaussian similiarity measure was used.

$$K(x_i, x_j) = \exp\left(\frac{-||x_i - x_j||^2}{2\sigma^2}\right), \tag{2.4}$$

where σ is a parameter to be chosen manually. It is refered at this point to intuitive explanation that if the two points are close to each other they must be really strongly related. This similarity transformation reduces the dimensionality of the data and pre-clusters the data into orthogonal dimensions. The Gaussian transformation (2.4) was used to construct the matrix A, then its diagonal is set to zero $diag(A) = 0$.

As a next step a matrix of degree D was constructed such that $D_{ii} = \sum_j A_{ji}$. Nextly, matrix D to construct Laplacian Graph $L = D^{-1/2}AD^{-1/2}$ is created. This Laplacian is positive-definite so it can be decomposed into $n = \dim(L)$ different eigenvalues and corresponding eigenvectors. The k largest eigenvectors to construct new matrix Y are used on which different clustering methods are tried.

2.5.3 Advantages and disadvantages

It should be noted that universality of spectral clustering is based on the fact that it does not make any assumptions on the form of the clusters as opposed to for example k-means method, where the resulting clusters are always convex sets.

2.6 Methods for identification the number of clusters

2.6.1 Partitioning around medoids

Next way to determine number of clusters K is partitioning around medoids (abbreviated further as PAM) as in [**?**].

The clustering procedure PAM ([9]) takes as an input a dissimiliarity matrix D and produces as an output a set of cluster centers (or so called "medoids"). In our case matrix of dissimiliarities is matrix A. Let K be the number of clusters and let $M = (M_1, \ldots, M_K)$ denote any size K collection of n elements x_j. One can then calculate dissimiliarity $d(x_j, M_k)$ of each element and each member of M.

Average silhouette method

One can consider K as given or it can be data-adaptively selected, for example, by maximizing the *average silhouette* as recommended by Kaufman and Rousseeuw [9]. Lets denote

$$d_1(x_j, M) = \min_{k=1,\ldots,K} d(x_j, M_k), \qquad l_1(x_j, M) = \left(\min_{k=1,\ldots,K} d(x_j, M_k)\right)^{-1},$$

where $d(x_j, M_k)$ is a dissimiliarty between each element and each member of M. The silhouette for a given element is calculated as follows:

1. For each data j calculate a_j which is the average dissimilarity of data j with other elements of its cluster:
$$a_j = \text{avg } d(x_j, x_{j'}),\ j \in \{i : l_1(x_i, M) = l_1(x_j, M)\}.$$

2. For each data j and each cluster k to which it does not belong calculate b_{jk} which is the average dissimilarity of data j with the members of cluster k:
$$b_{jk} = \text{avg } d(x_j, x_{j'}),\ j' \in \{i : l_1(x_i, M) = k\}.$$

3. Let $b_j = \min_k b_{jk}$. The silhouette of data j is defined by
$$S_j(M) = \frac{b_j - a_j}{\max(a_j, b_j)}. \tag{2.5}$$

It should be noted that value of silhouette index (2.5) is in range $[-1, 1]$. Values close to 1 indicates that the sample is far away from neighouring clusters, the values close to 0 indicates that the sample is close or on the decision boundary between two neighbouring clusters and values close to -1 suggests that those samples might have been assigned to the wrong clusters.

2.6.2 Calinsky criterion

The formula is
$$\text{Calinski Criterium} = \frac{SS_B/(k-1)}{SS_W/(N-k)}, \tag{2.6}$$
where SS_B and SS_W are respectively overall between and within cluster sum of squares, N is total number of observations and k is the number of clusters. For the optimal clustering the Calinski Criterium index should be the biggest possible. Following values for the Calinski index were obtained:

Table 2.1: Values of Calinski criterium index altogether with SSE

$k =$	1	2	3	4	5	6	7	8	9	10
SSE	80.24	77.56	75.06	72.75	70.60	68.44	66.56	64.80	63.00	61.26
calinski	NA	2.72	2.69	2.64	2.59	2.58	2.53	2.48	2.46	2.44

Values presented in the table 2.1 suggest that the optimal number of cluster is $k = 2$. Graphical results are presented on Figure **??**. We run this algorithm on both non-transformed data and matrix of dissimilarities of data. The results were similar - both methods indicated the same optimal number of clusters.

2.6.3 Affinity propagation

Another algorithm which was tried was *affinity propagation*[1]. The idea of the algorithm is based on "message passing" between the points. It will be shortly presented.

Affinity propagation is basing on finding *exemplars*, i.e. a subset of representative examples of given data. The algorithm simultanously considers all the data point as potential exemplars. By viewing each data point as a node in a network the algorithm recursively transmits real-value messages along the edges of the network until a good set of exemplers and corresponding clusters emerges. At any moment the magnitude of each message reflects the current affinity that one data point has for choosing another data point as its exempler, hence the method was called "affinity propagation".

Let $x_1, \ldots, x_k$ be a set of data points. In the algorithm we introduce a similiarity function s such that $s(x_i, x_j) > s(x_i, x_k)$ if and only if x_j and x_i are closer correlated than x_k with x_i.

[1]This part was prepared basing on [10].

In this case $s(x_i, x_k) = -||x_i - x_k||^2$. Affinity propagation takes as an input a real number $s(k,k)$ for each data point k so that data points with larger values of $s(k,k)$ are more likely to be chosen as an exemplar - they are refered as preferences.

They are two kinds of messages exchanged between data points and each takes into account a different kind of competition. The "responsability" $r(i,k)$ sent from data point i to candidate exemplar point k is to serve as the exemplar for point i taking into account other potential exemplars for point i. The "availability" $a(i,k)$ sent from candidate point k to point i reflects the accumulated evidence for how appriopriate it would be for point i to choose point k as its exemplar. At the beginning the availabilities are initialized to zero: $a(i,k) = 0$. The responsabilities are computed using the rule

$$r(i,k) = s(i,k) - \max_{k' \neq k}\{a(i,k') + s(i,k')\}.$$

As can be seen, in the first iteration $r(i,k)$ are set to the input similarity between point i and point k as its exemplar minus the largest of the similiarities between point i and other candidate exemplars.

Above responsibility update lets all candidates exemplars compete for ownership of a data point, the following availability update gathers evidence from data points as to whether each candidate exemplar would make a good exemplar:

$$a(i,k) = \min\left\{0, r(k,k) + \sum_{i' \neq \{i,k\}} \max\{0, r(i',k)\}\right\}$$

Only a positive portions of incoming responsabilities are added, because it is only necessary for a good exemplar to explain some data points well regardless of how poorly it explains other data points. If the self responsibility $r(k,k)$ is negative then that means that point k is better suited as belonging to another exemplar rather than being an exemplar itself.

For a point i the value of k that maximizes $a(i,k) + r(i,k)$ either identifies point i as an exemplar if $k = i$ or identifies the data point as an exemplar for point i.

2.6.4 Davies–Bouldin index

The Davies–Bouldin index (DBI) is a metric for evaluating clustering algorithms.

Let $R_{i,j}$ be a measure of how good the clustering scheme is. This measure, by definition has to account for $M_{i,j}$ the separation between the i-th and the j-th cluster, which ideally has to be as large as possible, and S_i, the within cluster scatter for cluster i, which has to be as low as possible. Hence the Davies–Bouldin index is defined as the ratio of S_i and M_i:

$$R_{i,j} = \frac{S_i + S_j}{M_{i,j}} \tag{2.7}$$

With this formulation, the lower the value, the better the separation of the clusters and the 'tightness' inside the clusters.

This is used to define Di:

$$D_i = \max_{i \neq j} R_{i,j} \tag{2.8}$$

If N is the number of clusters:

$$DB = \frac{1}{N}\sum_{i=1}^{N} D_i \tag{2.9}$$

DB is called the Davies–Bouldin index. This is dependent both on the data as well as the algorithm. Di chooses the worst-case scenario, and this value is equal to Ri,j for the most

similar cluster to cluster i. There could be many variations to this formulation, like choosing the average of the cluster similarity, weighted average and so on.

These conditions constrain the index so defined to be symmetric and non-negative. Due to the way it is defined, as a function of the ratio of the within cluster scatter, to the between cluster separation, a lower value will mean that the clustering is better. It happens to be the average similarity between each cluster and its most similar one, averaged over all the clusters, where the similarity is defined as Si above. This affirms the idea that no cluster has to be similar to another, and hence the best clustering scheme essentially minimizes the Davies–Bouldin index. This index thus defined is an average over all the i clusters, and hence a good measure of deciding how many clusters actually exists in the data is to plot it against the number of clusters it is calculated over. The number i for which this value is the lowest is a good measure of the number of clusters the data could be ideally classified into.

2.6.5 The gap statistic

The gap statistic is a method to estimate the number of clusters on a dataset, the concept is to compare the change within an output of a clustering algorithm such as k-means of hierarchical clustering, with a null reference distribution of the data.

$$\mathrm{Gap}_n(k) = E_n^*\{\log W_k\} - \log W_k \tag{2.10}$$

The concept of gap statistic is that in order to find an optimal number of clusters, we need to compare the graph the $logW_k$,where W_k is the error measure, with the null reference distribution of data, which refers to a distribution without an obvious clustering. The optimal number of clusters is the number that makes $logW_k$ falls the farthest bellow the reference curve. In the other case where the clusters are not well-defined, the goal will be to balance the maximization of the gap statistic. Tihshirani suggests the 1-standard-error method.

The method is to choose the cluster size $\hat{k}$ to be the smallest k such that:

$$Gap(k) \geq Gap(k+1) + s_{k+1} \tag{2.11}$$

In another words, we choose the number of clusters where the increase of the gap statistic starts to slow down.

2.6.6 The slope heuristic

Myself, this subsection is waiting for you!

Chapter 3

Clustering of Electricity Load Consumption

3.1 Data description

Within the framework of our project was proposed daily electricity load consumption data. It contains 70 observations with the following information:

- A signal vector of size 48 that corresponds to measurements of electricity consumption 2 times per hour during a day (i.e. $48 = 2 \cdot 24$).
- A number of a day in the week. Days are enumerated by numbers $1, 2, \ldots, 7$.

It is worth to mention that the data contains aggregated load by all consumers under analysis. In fact, this gives a great property. While load consumption of one user oscillates vastly, the aggregated one has fairly smooth shape. This fact allows us to work with the data as of the functional nature.

We noticed that days in our dataset do not correspond to real days of a week. It means 1 does not implies that a sample is related with Monday, and 2 corresponds to Tuesday etc. Indeed, we can split the data by days and plot a graph for each of them (Figure 3.1). Each curve represents 48 measurements during a day (for the sake of illustration, points were connected).

From the plots we can observe that subsets for days with numbers 3 and 4 have less consumption than the others. Actually, we can take into account the fact that most of factories and industries are closed on a weekend. From that we can deduce that consumption during a weekend is less compared to week days. Therefore, in our experiments we assume that third and fourth plots correspond to weekends. Because of that, we reinitialize days by shifting labels so that 1 will be assigned for Thursday, 2 is related with Friday and so on.

As it was described in 1.1, there are two approaches to work with the discretized data. The first one is to analyze directly the raw data of measurements. In the second case we represent observations via a basis expansion, then using it approximate curves, and finally treats the matrix of expansion coefficients as a new data that will be analyzed. As it was discussed in [3], the second approach is more preferable.

That is why, before direct data clustering, we approximate observations by B-splines. We use fairly a high degree of smoothing, which is 30. Then, to make a more compact representation we apply functional principal component analysis. In fact, with first two principal components 94.7% of the original variability is explained (Figure **??**). Therefore, we decide to reduce a dimension to 2D. Particularly, it helps to resolve the curse of dimensionality that arises when the number of estimated parameters is comparable with the number of observations. This leads to inadequate estimation. In our case, from the curse the Gaussian mixture model suffers the most. In fact, without FPCA it is even not possible to use this model.

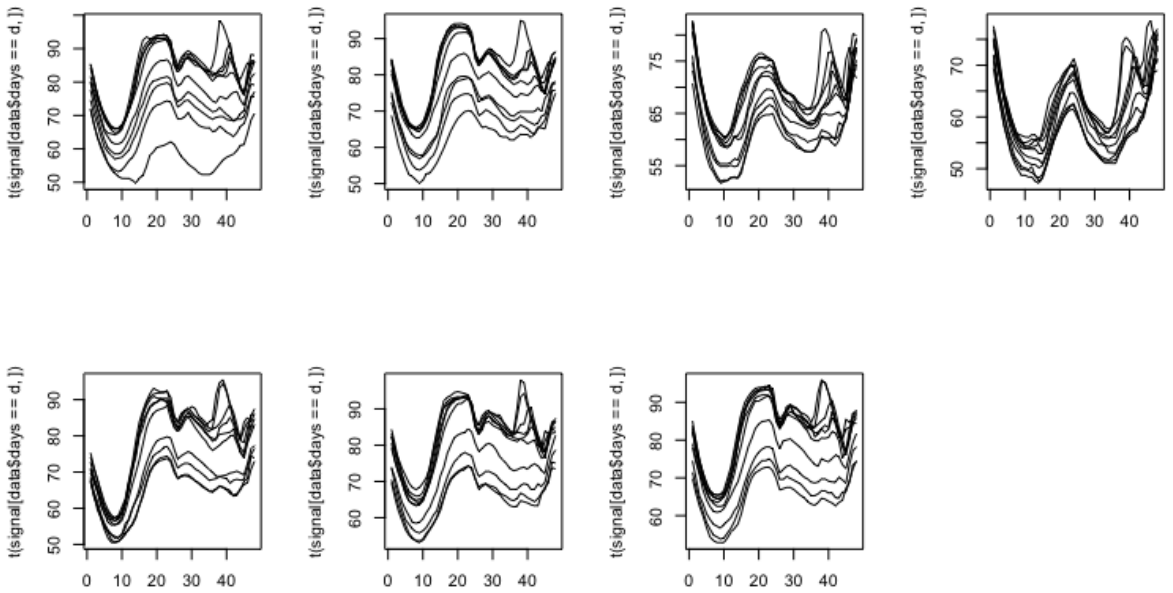
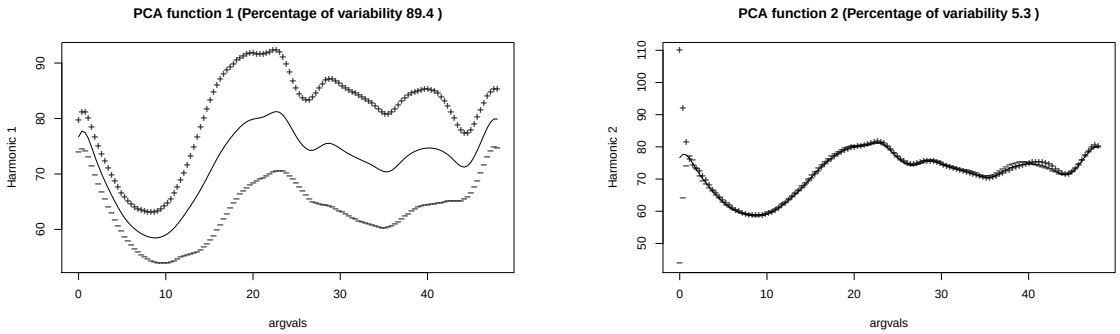


Figure 3.1



(a) The first principal component

(b) The second principal component

Figure 3.2: The graph of FPCA eigenfunctions.

3.2 Clustering data

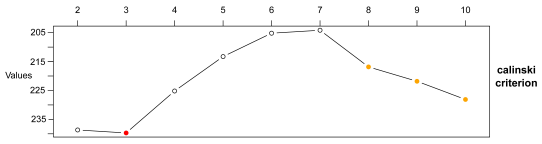
3.2.1 Introduction

In this section are presented the results of implementations different methods for evaluating efficient number of clusters such as Davies-Bouldin index (DBI), Calinski, GAP statistic, average silhouette,affinity propagation. Two of the listed methods (DBI and GAP) depends on the type of clustering algorithm, others are more primitive and depends just on the nature of data. Also here are covered clustering algorithms implementation and their results.

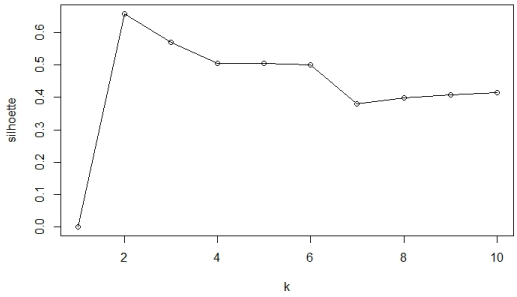
3.2.2 Estimation of the best number of clusters

Internal evaluation

Here, we apply such algorithms as Calinski criterion, affinity propagation, average silhouette.



(a) Calinski criterion.



(b) Average silhouette.

Figure 3.3: Results of applying Calinski criterion and average silhouette. The former one shows that the number of clusters is 3, while the latter one has maximum value for 2 clusters.

Average silhouette shows how tightly grouped the data in the clusters. Hence, it is needed to choose a number of clusters with the maximum silhouette. In the figure we can observe that 2 is an optimal number of clusters. Conversely, Calinski criterion chooses the minimum value, which in our case is 3. Last criterion is affinity propagation. The result of this is 5 clusters.

Davies-Bouldin index for k-means++

There were carried out 5 experiments by k-means++ clustering method for different number of clusters and for each experiment was calculated Davies-Bouldin index. Below is shown graph of the dependence between Dunn index and the numbers of clusters:

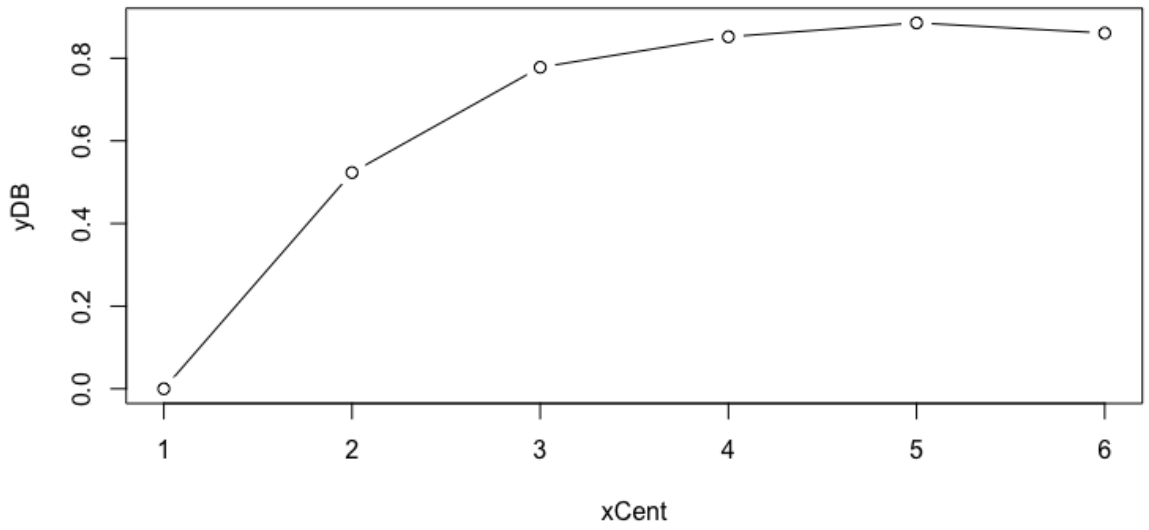


Figure 3.4: Results of applying Davies–Bouldin index for k-means.

As was described in 2.6.4 section, the lowest value of DB index means a good measure of the number of clusters the data could be ideally classified into. In this case the best numbers of clusters (centroids) are **2**.

GAP statistic for k-means++

The main idea of this approach is to obtain an ideal clustering, you should select k (where k is number of centroids) such that you maximize the gap statistic. But in our case the clusters are not as well-defined, and we want to be able to balance maximizing the gap statistic with parsimony of the model. Assuming that that plot is just going to continue to increase, of course, the results are less useful. So Tibshirani suggests the 1-standard-error method:

Choose the cluster size $\hat{k}$ to be the smallest k such that:

$$Gap(k) \geq Gap(k+1) + s_{k+1} \tag{3.1}$$

Which informally is identifying the point at which the rate of increase of the gap statistic begins to "slow down". So, in presented below figure, point corresponds to **k=2** satisfy this condition.

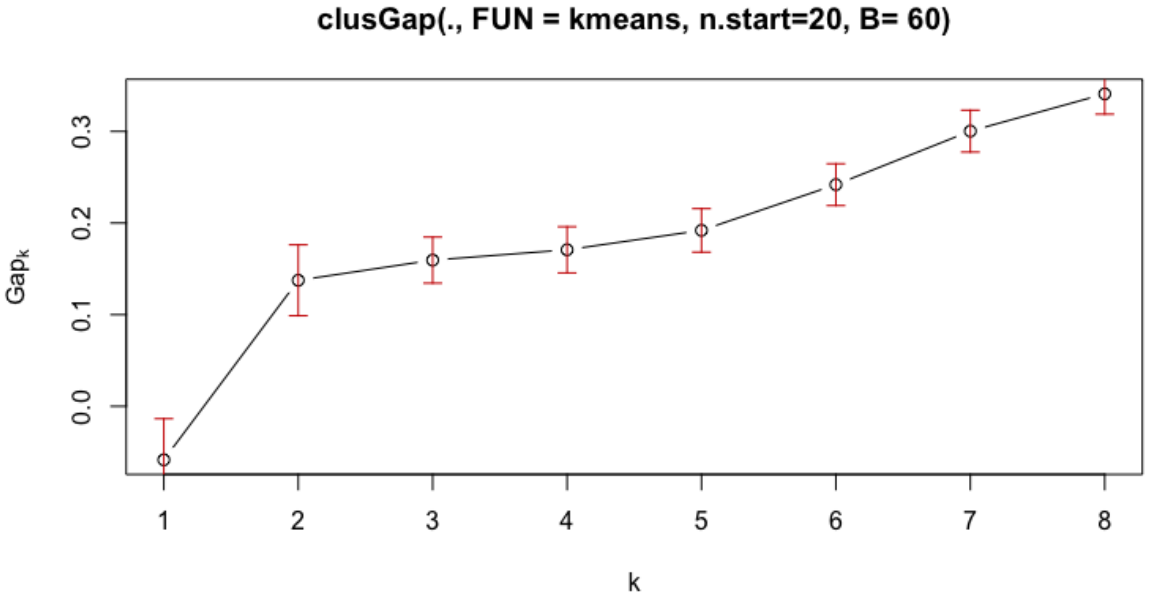


Figure 3.5: Results of applying GAP statistic for k-means.

Hierarchical clustering

Then, we can determine the number of cluster for hierarchical clustering in each case: average, complete and single linkage. First, figure demonstrates the result of applying Davies-Bouldin index.

As it can be seen, the same results take place for all the types of linkage, which is 2. After that, the gap statistic can be applied (figure 3.8).

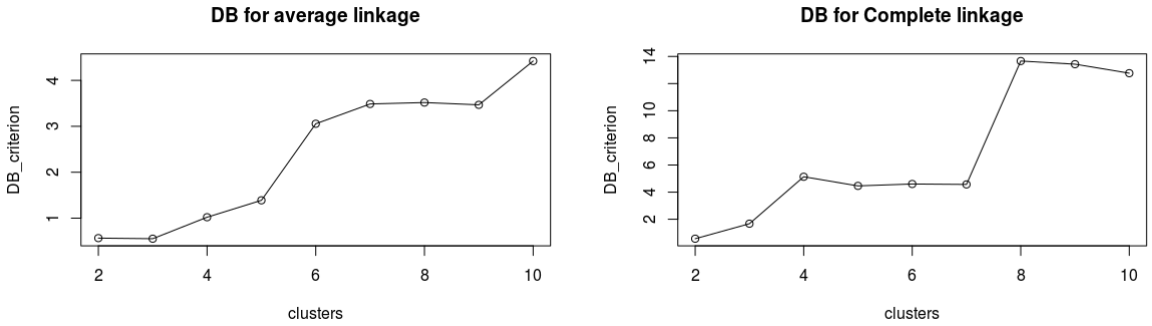


Figure 3.6: Davies Bouldin index for Hierarchical clustering in case of average linkage(left), and complete linkage (right)

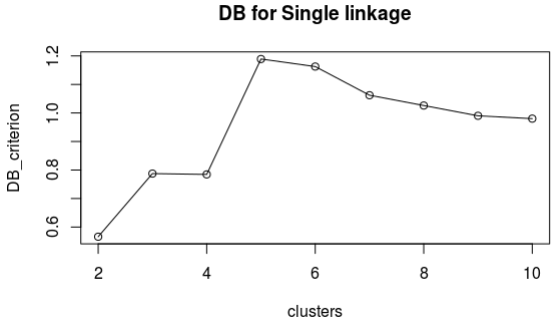


Figure 3.7: Davies Bouldin index for Hierarchical clustering in case of single linkage

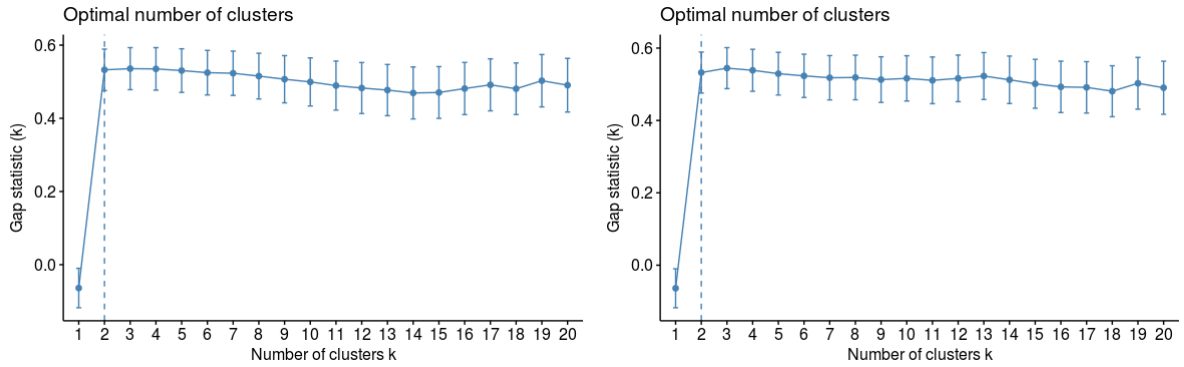


Figure 3.8: Gap statistic for hierarchical clustering in the case of complete linkage(left), and average linkage (right).

For both complete and average linkage, the optimal number of clusters is 2, since the optimal number of clusters using gap statistic is the value where the increase of the gap statistic starts to slow down, in our case it is number 2 for both linkage.

In addition, we can deduce the number of clusters using dendograms that are depicted in the figure 3.10.

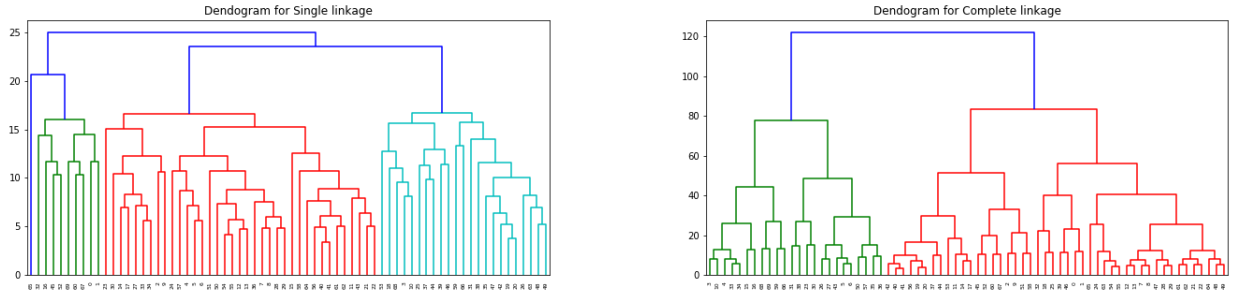


Figure 3.9: Dendogram in case of single linkage (left) and complete linkage (right)

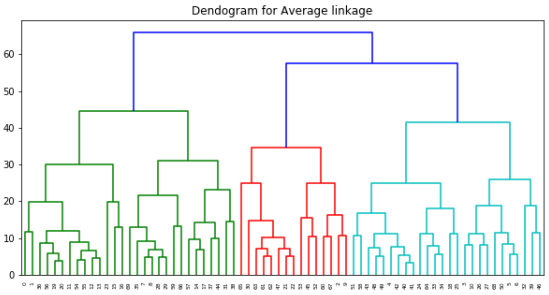
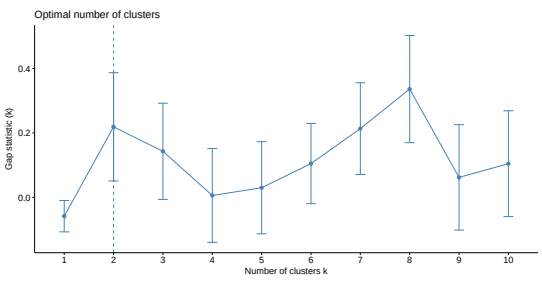


Figure 3.10: Dendogram in case of Average linkage

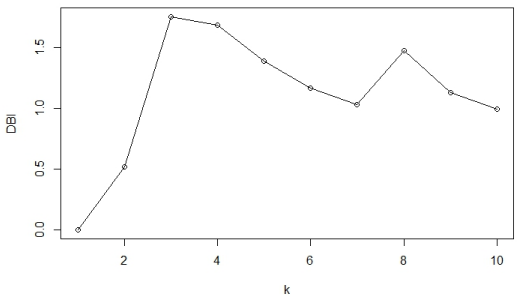
The three dendograms illustrate the different types of linkage, it is clear that for the method Average and Complete linkage, the number of clusters is two, however the number of clusters for the single cluster is four. Although the dendogram gives an idea about the different clustering levels, it can be sometimes hard to specify a certain number of clusters. Generally the good dendogram like the complete average shows a big distance between two clusters.

The EM algorithm

Now, let's move to the determining the optimal number of clusters for the EM algorithm. On the figure 3.11 we can observe the results for the Gap statistic and Davies-Bouldin index. Cases from 1 to 10 clusters were tested. We observe that 3



(a) The gap statistic plot.



(b) The plot of DBI for different number of clusters.

Figure 3.11: Results of applying the Gap statistic and the Davies-Bouldin index for the EM algorithm. Both show that the optimal number is 2.

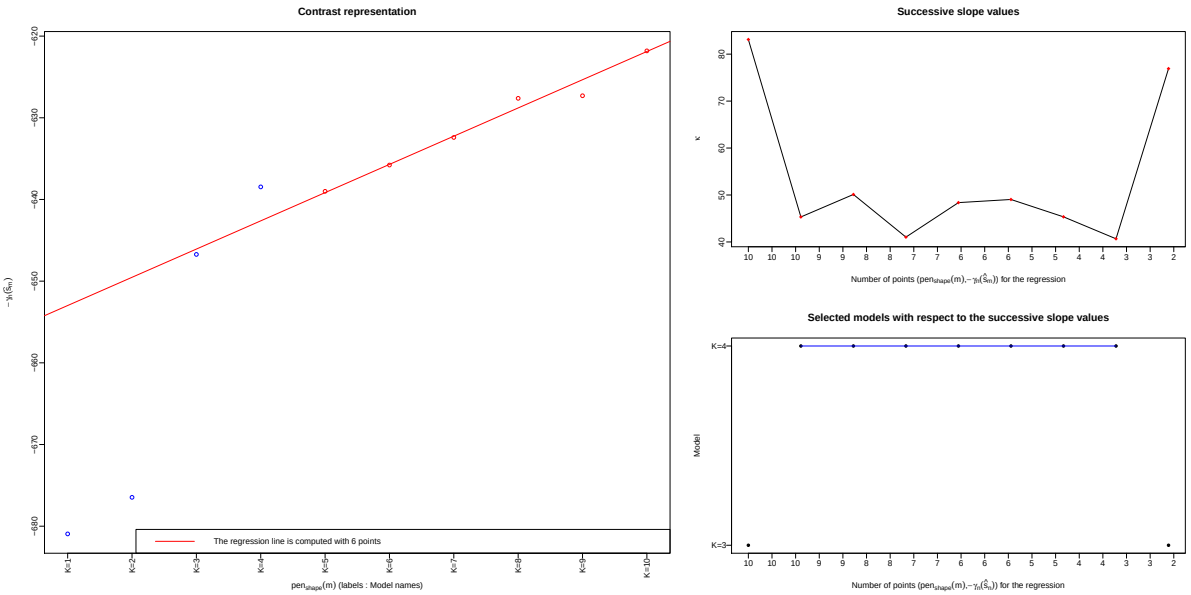


Figure 3.12: Results of applying the slope heuristic. It estimates the number of clusters with 4.

Spectral Clustering

This subsections contains results and indicated number of groups obtained via running algorithms described previously on transformed data as in 2.5.1.

It should be mentioned that this part was prepared basing on the original data which were directly transformed via Gaussian kernel as described in section 2.5.1 and later the remaining of the algorithm was applied. Meanwhile, other parts were prepared basing on projected data. The reasons of differencing here were following:

1. firstly, the algorithm stated as in 2.5.1 needs a lot of data point to run effectively,
2. secondly, one may doubt in sense of clustering data which were prepared using two uncorrelated transformations - the structure of those may not be preserved.

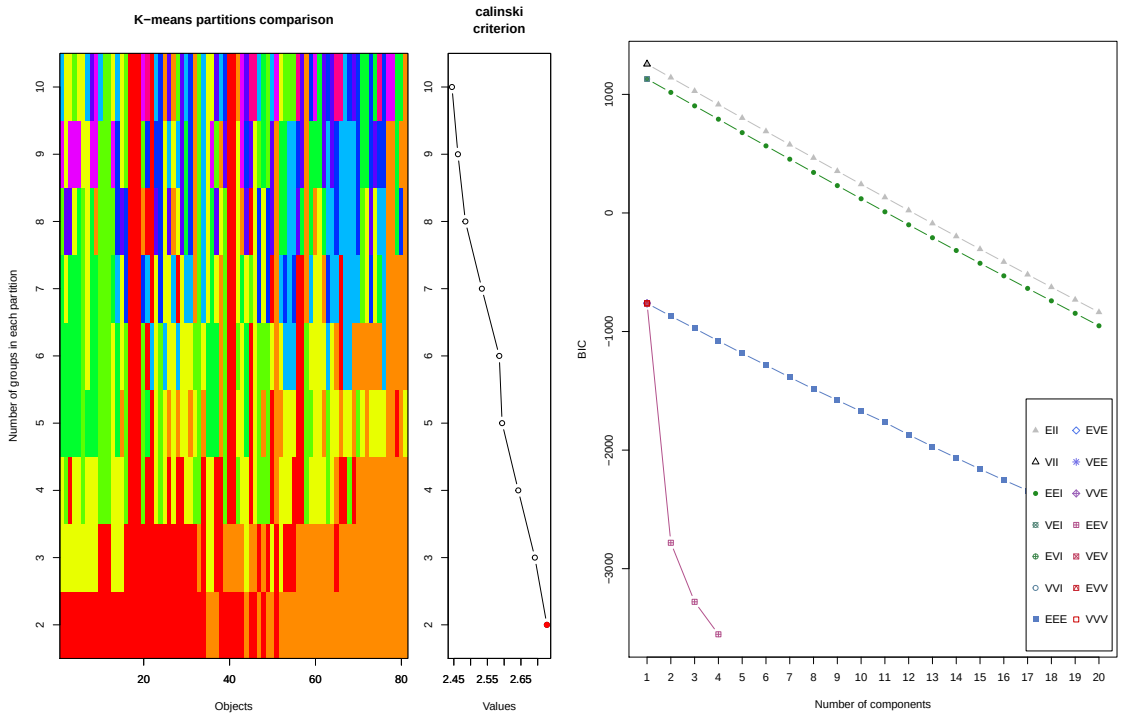


Figure 3.13: Calinski index values on transformed data (left) and results of Bayesian Information Criterium values used to determine proper number of groups on transformed data (right).

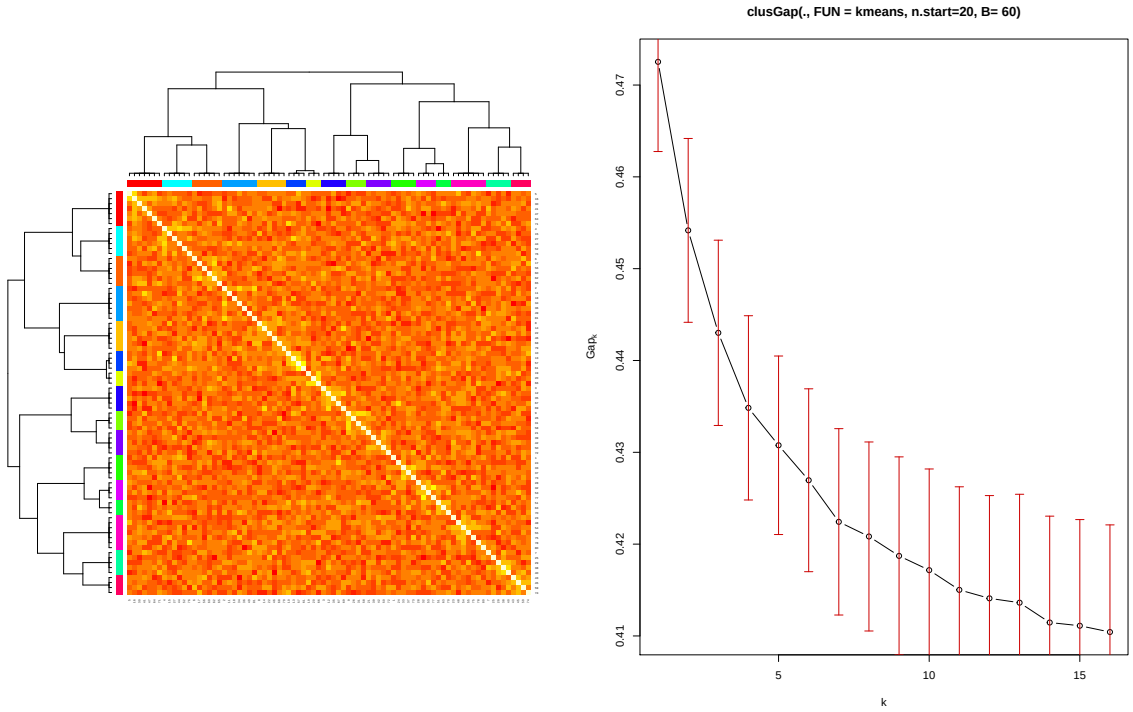


Figure 3.14: Affinity propagation method with resulting heatmap. As can be seen, this results is particularly inconsistent with previously received (left) and Spectral gap clustering results on transformed data (right).

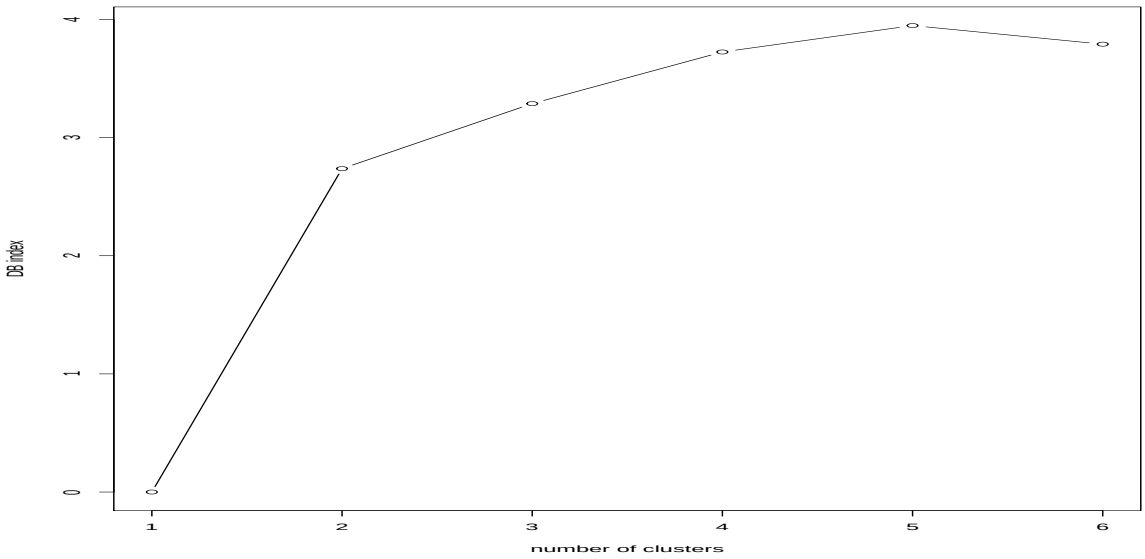


Figure 3.15: DB index values obtained from nonlinear k-means clustering

As one may notice, not all results all consistent when performed on transformed data, however the small number of clusters is in favor. It may be suggested to treat the results of affinity propagation as an outlier, it was decided to leave it in educational purposes. There is one method which may be still worth giving a try here, namely Davies-Bouldin index for Hierarchical clustering. Having done this one may try averaging all the methods run on transformed data, trying different clustering models on projected data and selecting one according to chosen criteria.

That being said, on the figure 3.15 is included the plot of DB index values which resulted after running k-mean on transformed data. According to previously stated theoretical explanation, it is justified to choose the proper clustering number as the one with lowest value of index. It is possible to see that the most proper clustering seems to be the one with just 2 clusters.

3.3 Clustering algorithm implementation

3.3.1 Introduction

In 3.2. section was shown that the most efficient number of clusters for each method is 2 (expect of EM algorithm?). In this chapter it presents results of clustering algorithms implementation such as k-means++, EM, hierarchy, spectral with chosen number of clusters k=2.

3.3.2 k-means++

R program language was used to write k-means++ function that allows as to clusterize initial data set using the main idea is iteratively find such centroids that should to minimize the within-cluster sum of square. Below is shown this function:

To validate this function it was decided to build 1-d plot with average means of each sample and display there found centroids.

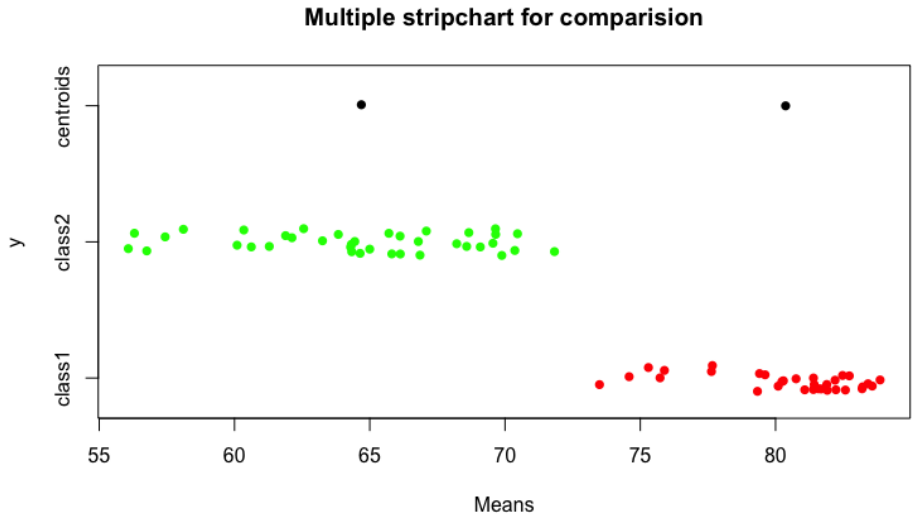


Figure 3.16: Validating k-means++ function

Since the main idea of k-means method is finding center of mass for each cluster,so it was not surprised that after deployment of k-means method to changed signal dataset, It was divided into classes that correspond to the average values of electricity consumption. It means in the case of k = 2, we obtained 2 classes corresponding to the average value for low and high electricity consumption.

Applying k-mens to the prepared dataset, we observe the same tendency that was in validation result .

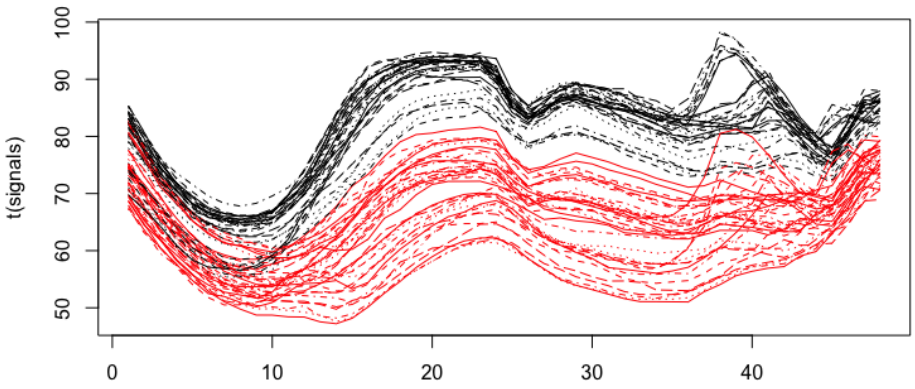


Figure 3.17: Result of clustering using k-means++ function

It was also noticed that the weekends don't belong to the 1 cluster. It means that in the weekends it spends less electricity than in working days.

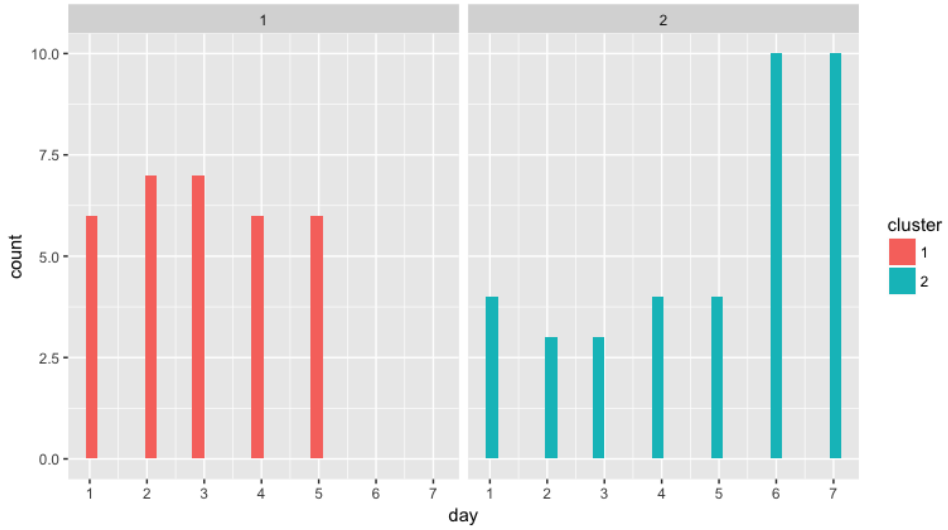


Figure 3.18: Bar chart days per cluster

Below presented average mean of electricity consumption per day. It was noticed that the consumption in Saturday does not exceed 70, whereas Sunday 66.

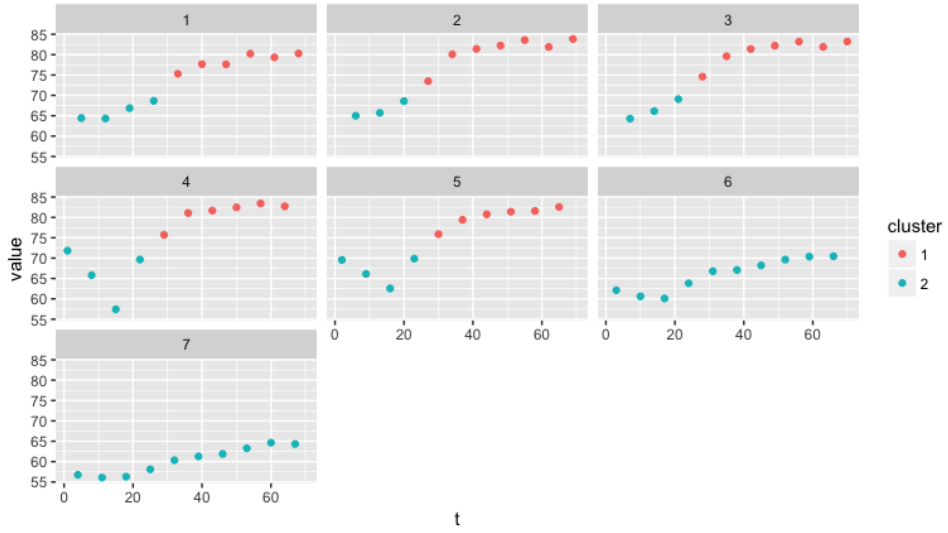
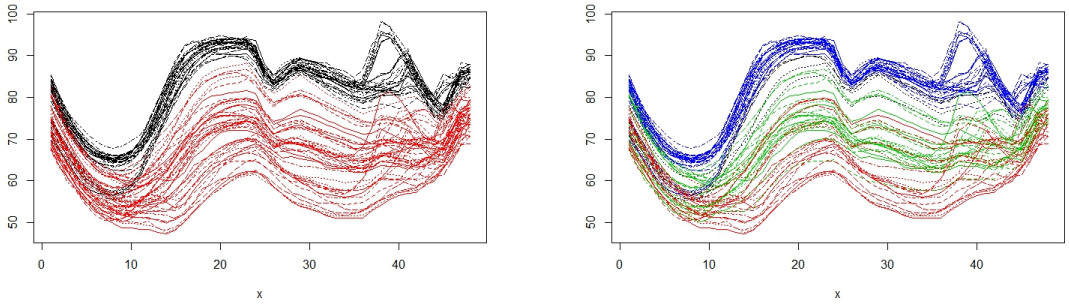


Figure 3.19: Average mean of electricity consumption per day

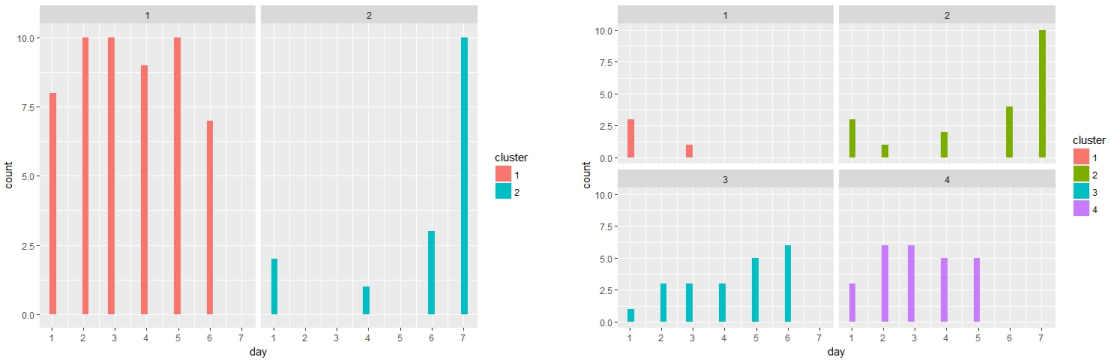
3.3.3 The EM algorithm

The EM algorithm for a Gaussian mixture model has been implemented by R language. It has two cases, namely, with full covariance matrices and with diagonal ones. The code can be found in appendix. Results from the paragraph 3.2.2 show that the number of clusters can be either 2 or 4 clusters. The plot of original data with labels are illustrated in the figure 3.20. We can see that one cluster is clearly recognizable for both cases. More exactly, it is a cluster of high consumption days (The black color for (a) and the blue color for (b)). Then we can plot histograms of data distributed by days (figure 3.21). For both cases Sunday observations are belong to clusters with low consumption. This also can be observed in the figure 3.22.



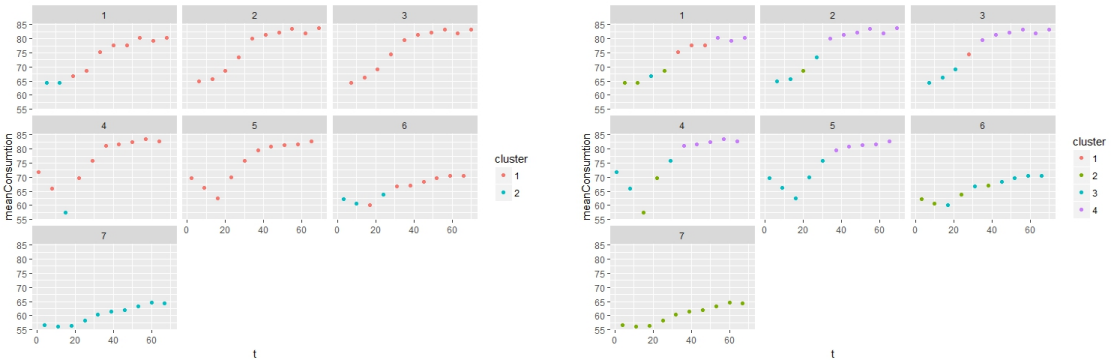
(a) 2 clusters. (b) 4 clusters.

Figure 3.20: Original curves painted by cluster labels.



(a) 2 clusters. (b) 4 clusters.

Figure 3.21: Observations are split by days and cluster labels.



(a) 2 clusters. (b) 4 clusters.

Figure 3.22: Average consumption per day painted by cluster labels and split by days.

3.3.4 Hierarchical clustering

Python language was used to write hierarchical clustering, using several functions from calculating euclidean distances till the hierarchical clustering function itself, it was possible to cluster the data into all level of clusters and according to three types of linkage. The function of the algorithm in on the appendix

The number of clusters that was agreed on according to the previous chapter was 2 for the the average and complete linkage.Hierarchical clustering using single linkage didn't show a good

cut on the dendogram, and it also didn't show a good difference between four clusters retained by the dendogram cut, or two clusters that were agreed on.

When inspecting the contains of the clusters, it didn't show any pattern in days, so the next step was to compare the mean of each cluster with another, to refer to winter or cold days when the electricity consumption is very high during the day, and the consumption during summer and hot days.

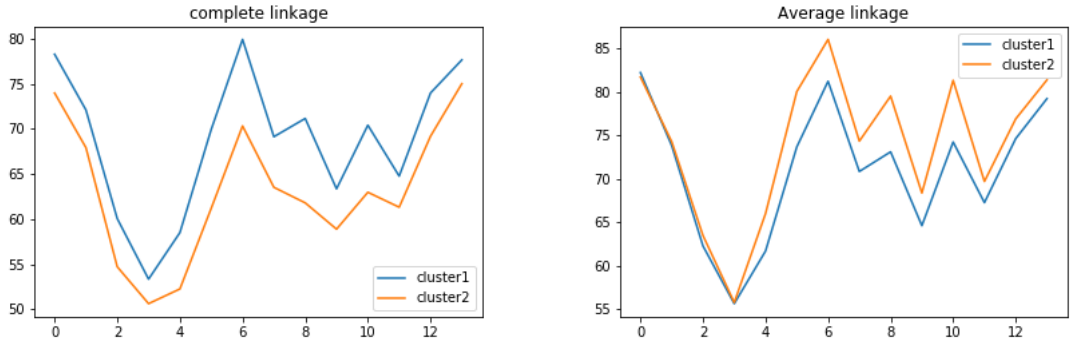


Figure 3.23: Electricity consumption during the day for the two clusters output of hierarchical clustering using Complete and Average linkage

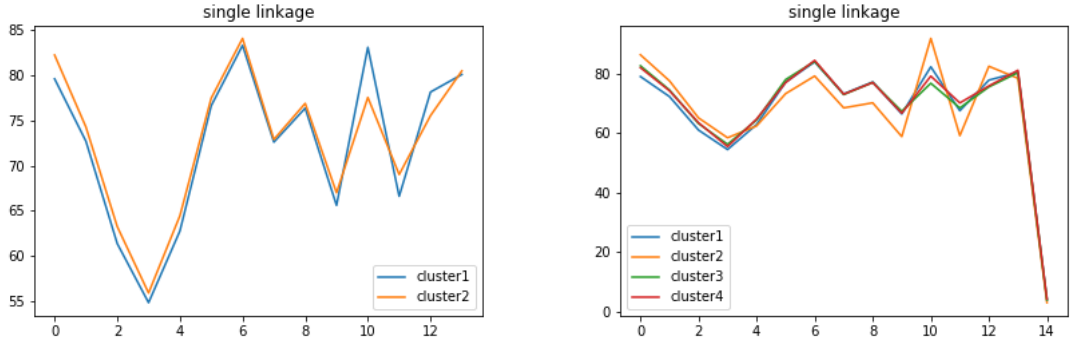


Figure 3.24: Electricity consumption during the day for the two then three clusters output of hierarchical clustering using Single linkage

the previous plots show that there was some significant difference between the clusters in complete linkage, and between clusters of average linkage during a certain time of the day.

Conclusion

In this work hierarchical clustering was applied on daily electricity load consumption approximated data, three methods of linkage to merge clusters were used, single, complete and average linkage. The clustering was done with respect to these methods, and showed some differences of the output of the algorithm.

When applying gap statistic and Davies Bouldin index to find the optimal clustering, two clusters was the output for hierarchical clustering, especially when using complete and average linkage.

In addition, another way to decide the optimal number of clusters was with dendograms that are special to hierarchical algorithm, and that confirmed two clusters as the optimal number of clusters for hierarchical clustering with complete and average linkage, while it suggested one more for the single linkage.

Now that the decided number of clusters is two, we went back and cut the hierarchical tree to get the two clusters and three for the single linkage. Plotting the clusters in function of the days didn't make much sense. Only when calculating and plotting the mean of each cluster in ranges of the day, that we found a difference between clusters, that reflect a pattern of season change.

So one cluster probably refers to a period where the electricity is highly consumed during the day which is during winter, where heating works all the day, and the other cluster refers to the hot days, where the electricity is lower.

3.3.5 Spectral clustering

Several methods to obtain the right number of clusters were tested in this part. The unique answer about the appropriate number of clusters was not obtained, however usually it was small number, most often two clusters were indicated. It is worth to note that the methods were tested not for the original data, but the one transformed under Gaussian kernel and prepared as described in section 2.5.1.

Below we will shortly state results obtained with each method. Method called average silhouette as described in 2.6.1 indicates $k = 4$ clusters as the most adequate clustering. Further, second method, based on the value of Calinsky Index as introduced in section 2.6.2 indicates $k = 2$ clusters as the most adequate partitioning. Finally, two last methods, i.e. information theory based criterium BIC as presented in section **??** and affinity propagation method (section 2.6.3) provides us with respectively $k = 1$ and $k = 16$ groups of data.

Several methods considered helpful in indicating proper number of groups in data were tested. Each of them performed differently on the transformed data, however the tendency was to keep with low number of clusters. One may reach conclusion that the result is highly inconsistent. To elaborate, it is advised to try different transformations of data, try different variance in Gaussian kernel used, try different number of eigenvectors with which new data matrix was constructed. Further, different models with each indicated number of groups may be tested and investigated. Next issue, which is however believed to be of little importance is that not all data was used, i.e. 6561 out of 6720 records available were used. It is a consequence of the algorithm, which demands a quadratic matrix. It is not likely to heavily influence the results, however it might have some impact on the lack of consistency.

Conclusion

In this work paper was provided initial data set of electricity load consumption. To explore and analyze ones, it was decided implement different clustering algorithms. For correct and efficient clustering it was carrying out series of experiments aim to evaluate the best number of clusters. In experiments were involved such evaluating methods as Davies–Bouldin index, GAP statistic, Calinski, affinity propagation and average silhouette. The results were provided below (Table 3.1).

Number of clusters for each linkage	DB	GAP	Calinski	affinity propagation	average silhouette
K-means++	2	2	3	5	2
Hierarchical	2	2	3	5	2
EM	2	2	3	5	2
Spectral clustering	2	2	3	5	2

Table 3.1: Table of the number of clusters for each linkage

The results of clustering were shown that data set was clusterized on 2 clusters of high and low mean of electricity load consumption. The cluster that corresponds to high load consumption doesn't consist of weekends day.

Bibliography

[1] Silverman B. W. Ramsay J. O. *Applied Functional Data Analysis: Methods and Case Studies*. Springer-Verlag New York, 2002.

[2] Graves S. Ramsay J. O., Hooker G. *Functional Data Analysis with R and MATLAB*. Springer-Verlag New York, 1st edition, 2009.

[3] Julien Jacques and Cristian Preda. Functional data clustering: A survey. *Adv. Data Anal. Classif.*, 8(3):231–255, September 2014.

[4] Silverman B. W. Ramsay J. O. *Functional Data Analysis*. Springer-Verlag New York, 2nd edition, 2005.

[5] Benko M. Functional Data Analysis with Applications in Finance. `https://edoc.hu-berlin.de/bitstream/handle/18452/16237/benko.pdf?sequence=1`, 2006.

[6] Voroncov K. V. Lecture notes in statistical (bayesian) algorithms of classification. `http://www.machinelearning.ru/wiki/images/e/ed/Voron-ML-Bayes.pdf`, 2009.

[7] Nalisnick E. Smyth P. The EM Algorithm for Gaussian Mixtures. `http://www.ics.uci.edu/~smyth/courses/cs274/notes/EMnotes.pdf`, 2017.

[8] A.Ng et al. On spectral clustering: Analysis and an algorithm. 2009.

[9] P.J.Rousseeuw. L.Kaufman. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, New York, 1990.

[10] D.Dueck. B.J.Frey. Clustering by passing messages between data points. 2007.

Appendix

Here, implementation of clustering algorithms that are considered in this paper are presented.

K-means

```
kmc<-function (data,k){
  #initializing=====================================
    c<-1
    cent=matrix(0,nrow = 0,ncol = ncol(df))
    dist=matrix(0,nrow = nrow(df),ncol = k)
    prob=matrix(0,nrow = nrow(df),ncol = k)

  #k-means++ part of calculation mean of centroids===
    df_cent=df
    for(j in 1:k){
        cent=rbind(cent,df_cent[c,])
        for(i in 1:nrow(df_cent)){
          dist[i,j]=euclidean_distance(df[i,-1],cent[j,-1])
        }
        ss<-colSums(dist)
        for(m in 1:nrow(df_cent)){
          prob[m,j]=dist[m,j]/ss[j]
        }
        c=which.max(prob[,j])
    }

  #ordinary k-means=================================

  distkms=matrix(0,nrow=nrow(df),ncol = k)
  centc=cent
  eps=1e-5
  repeat{
    #=============================================
    clustvec=matrix(0,nrow=nrow(df),ncol=1)
    for (j in 1:k){
      for (i in 1:nrow(df)){
        distkms[i,j]=euclidean_distance(df[i,-1],cent[j,-1])
      }
    }
    #=============================================
    for (i in 1:nrow(df)){
      clustvec[i]=which.min(distkms[i,])
    }
    #=============================================
```

```
    cs=matrix(0,nrow=0,ncol=ncol(signals))

    for (i in 1:k){
      cs=rbind(cs,colSums(signals[which(clustvec %in% i),]))

    }
    #============================================
    for(i in 2:ncol(df)){
      for(j in 1:k){
        centc[j,i]=cs[i-1]/length(which(clustvec %in% j))
      }
    }


    RsNew=rowSums(centc)
    RsOld=rowSums(cent)
    print(cent[,2])
    print(centc[,2])
    print(RsOld)
    print(RsNew)


    if(sum(setdiff(RsNew,RsOld))<=eps){
      break
    }else{cent=centc}

  }
}
```

hierarchical clustering

```
def find_clusters(input_matrix,linkage):
# first step is to find the minimum value of a distance matrix
    for p in range(1,matrix.shape[0]):
        min_val = sys.maxsize
        for i in range(0, matrix.shape[0]):
            for j in range(0, matrix.shape[1]):
                if(matrix[i][j]<=min_val):
                    min_val = matrix[i][j]
                    row_index = i
                    col_index = j
# Now is the step to update the matrix
# several linkage methods are used to calculate the new distance between clusters
# case of single linkage
        if(linkage == "single" or linkage =="Single"):
            for i in range(0,matrix.shape[0]):
                if(i != col_index):
                    temp = min(matrix[col_index][i],matrix[row_index][i])
                    matrix[col_index][i] = temp
                    matrix[i][col_index] = temp
# case of complete linkage
        elif(linkage=="Complete" or linkage == "complete"):
             for i in range(0,matrix.shape[0]):
```

```
                if(i != col_index and i!=row_index):
                    temp = max(matrix[col_index][i],matrix[row_index][i])
                    matrix[col_index][i] = temp
                    matrix[i][col_index] = temp
# case of average linkage
        elif(linkage=="Average" or linkage == "average"):
             for i in range(0,matrix.shape[0]):
                if(i != col_index and i!=row_index):
                    temp = (matrix[col_index][i]+matrix[row_index][i])
                    temp= temp/2
                    matrix[col_index][i] = temp
                    matrix[i][col_index] = temp
# setting the rows to infinity ( like deleting them)
        for i in range (0,matrix.shape[0]):
            matrix[row_index][i] = sys.maxsize
            matrix[i][row_index] = sys.maxsize
# the next step is keeping track of the clusters made by saving them in a dictionary
# this returns Z
    return Z
```

EM algorithm

```
#Auxiliary functions
#Computation the density function's value a given observation
Gaussian_Density <- function(x, mean = matrix(rep(0,length(x))),
                                  cov_matrix = diag(length(x))){
  p<-length(x)
  return(1/((2*pi)^(p/2)) * sqrt(1/det(cov_matrix)) *
  exp(-0.5 * t(x-mean) %*% solve(cov_matrix) %*% (x-mean)) )
}

#Convert to vector-column
as_vector <- function(x){
  return(t(as.matrix(x)))
}

#Log-likelihood
LLH <- function(data, means, covs, priors){
  value <- 0
  for(i in 1:length(data[,1])){
    element <- 0
    for(j in 1:length(means)){
      element <- element + priors[j] *
      Gaussian_Density(as_vector(data[i,]), mean = means[[j]],
                                       cov_matrix = covs[[j]])
    }
    value <- value + log(element)
  }
  return(value)
}
#-----------------------------------------------

#Implementaion of EM for Gausssian mixture model
```

```r
Gaussian_EM <- function(data, k=2, max_iter = 100, eps = 1e-05, diagonalized = F){
  n<-length(data[,1])
  p<-length(data[1,])

  #Initialization Step
  means <- vector("list", k)
  covs <- vector("list", k)

  num_of_points <- sample(1:n,k)
  if(diagonalized){
    pool_cov <- diag(sapply(1:p,function(j){var(data[,j])}))
    K <- k*(p+p+1)-1
  } else {
    pool_cov <- cov(data)
    K <- k*(p+p*(p+1)/2+1)-1
  }
  priors <- rep(1/k,k)
  for (i in 1:k){
    means[[i]] <- as_vector(data[num_of_points[i],])
    covs[[i]] <- pool_cov
  }
  loglh <- LLH(data,means,covs,priors)
  print("Initialization Step Is Done")

  #EM
  iter <- 1
  posteriors <- matrix(0, nrow = n, ncol = k)
  repeat{
    #Expectation
    for (i in 1:n){
      full_prob <- 0
      for(j in 1:k){
        posteriors[i,j] <- priors[j] *
        Gaussian_Density(as_vector(data[i,]), mean=means[[j]], cov_matrix=covs[[j]])
        full_prob <- full_prob + posteriors[i,j]
      }
      posteriors[i,] <- posteriors[i,] / full_prob
    }
    #Maximization
    for(j in 1:k){
      means[[j]] <- colSums( data*posteriors[,j] ) / sum(posteriors[,j])
      covs[[j]] <- matrix(0,nrow=p,ncol=p)
      for(i in 1:n){
        if(diagonalized){
          covs[[j]] <- covs[[j]] + diag(c((data[i,]-means[[j]])^2)) * posteriors[i,j]
        } else {
          covs[[j]] <- covs[[j]] +
          ((as_vector(data[i,])-means[[j]]) %*%
          t( as_vector(data[i,]) - means[[j]]) ) * posteriors[i,j]
        }
      }
      covs[[j]] <- covs[[j]] / sum(posteriors[,j])
```

```
      priors[j] <- sum(posteriors[,j])/n
    }
    #Condition checking
    old_loglh <- loglh
    loglh <- LLH(data,means,covs,priors)
    bic <- log(n)*K - 2*loglh
    if((abs(loglh-old_loglh)<eps)|(iter>=max_iter)) break
    iter <- iter + 1
  }
  return(list(means=means, covs=covs, priors=priors, iter=iter,
                            loglh = loglh, bic = bic))
}

#Prediction of labels
predict.Gaussian_EM <- function(data, model, prob = F){
  n<-length(data[,1])
  k<-length(model$priors)
  cluster <- numeric(n)
  posteriors <- matrix(0, nrow = n, ncol = k)
  for (i in 1:n){
    for(j in 1:k){
      posteriors[i,j] <- model$priors[j] *
      Gaussian_Density(as_vector(data[i,]), mean = model$means[[j]],
                                              cov_matrix = model$covs[[j]])
    }
    cluster[i] <- which(posteriors[i,] == max(posteriors[i,]))
  }
  if (prob){
    return(posteriors)
  } else {
    return(cluster)
  }
}
```

Spectral

For the spectral clustering mainly `R` programming language was used. In this section the algorithm described in 2.5.1 is written as a code.

```
M = matrix(signals[1:6561], nrow=81, ncol=81)

#1st step : form an affinity matrix
sigma = 1
#sigma above is to be changed and tried with different values
MatrixTransform = function(x){ exp( -x^2/2/sigma^2 ) }
A = apply(M-t(M),2,MatrixTransform)
diag(A) = 0

#2nd define a diagonal matrix whose ii-th element is a sum of a i-th row of a matrix A

SumRows = function(x){ sum(x) }
SumMatrix = apply(A, 1, SumRows)
```

```
InvertAndRootSquare = function(x){ x^{-1/2} }
Conv = lapply(SumMatrix,  InvertAndRootSquare)

D = diag(Conv, nrow = length(Conv), ncol = length(Conv))

#constructing new matrix L

L = D %*% A %*% D
L = zapsmall(L, digits = 8) # remove values close to 0

#3rd step - find k largest eigenvectors of a matrix

k = 30
#k is an arbitrary value, it is to be changed and tried with values ranging from 3 to 35
EigenL = eigen(L)
#choose k largest eigenvectors and write them to a new matrix in columns
EigenVectos = EigenL$vectors[1:k,]
X = EigenVectos

#4th form new matrix Y from X by normalizing each row of X to have an unit length
#(with respect to 2 norm)

norm<-function(x){return (x/sqrt(sum(x^2))) }
Y = t(apply(X, 2 , norm) )
```