Universidade Federal do Rio de Janeiro

Gabriel Castor de Azevedo

Cluster Analysis in High Dimensions

Rio de Janeiro
Junho 2017

Gabriel castor de Azevedo

Cluster Analysis in High Dimensions

Dissertação de mestrado apresentada ao Programa de Pós-graduação do Instituto de Matemática, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do grau de Mestre em Matemática.

Orientador: Fabio Tavares Ramos Doutor em Matemática - Universidade Federal do Rio de janeiro

Rio de Janeiro
Junho 2017

CIP - Catalogação na Publicação

d354c

de Azevedo, Gabriel Castor
Cluster Analysis in High Dimensions / Gabriel Castor de Azevedo. -- Rio de Janeiro, 2017.
149 f.

Orientador: Fabio Antônio Tavares Ramos.
Dissertação (mestrado) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Programa de Pós-Graduação em Matemática, 2017.

1. Cluster Analysis. 2. Unsupervised Learning. 3. High Dimensions. I. Ramos, Fabio Antônio Tavares, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a).

Gabriel Castor de Azevedo

Cluster Analysis in High Dimensions

Dissertação de mestrado apresentada ao Programa de Pós-graduação do Instituto de Matemática, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do grau de Mestre em Matemática.

Aprovada em

Presidente, Prof. Fabio Antônio Tavares Ramos, IM, UFRJ

Prof. Glaydston Mattos Ribeiro, PET, COPPE-UFRJ

Prof. Heudson Tosta Mirandola, IM, UFRJ

Prof. Yuri Fahham Saporito, EMAp, FGV-Rio

Rio de Janeiro
Junho 2017

This thesis is dedicated to my dear wife Laura.

This work was the result of a three year masters which would not have been possible without the support in every aspect of my dear wife Laura. Laura, I am grateful for your help, patience, insights, discussions and everything else we lived together all these years around this work. I might say we have met in an outlier event and we keep - for good - outside the cluster of normality.

I thank my advisor Fabio Ramos who introduced me to this new exciting world of Data Science and Machihne Learning. It opened new horizons and gave new perspectives. Thank you Fabio for pushing me forward by always chalenging me, giving me autonomy to think, making me face true mathematics through real world problems!

I give special thanks to Yuri Saporito who played a major role in my academic and professional development by giving me the opportunity to study two trimesters at Fundação Getúlio Vargas. There I have learnt all the programming tools I needed for this work, otherwise it wouldn't be possible.

I want to thank as well the good discussions I had with Heudson Mirandola. Though we did not encountered many times to talk about cluster analysis and our results (shame on me) I might say that many important insights were essential to this work and for my understanding of the topic.

It is extremely important to mention that the Departamento Nacional de Infraestrutura de Transportes (DNIT) allowed us to use the traffic count data of brazilian main roads, so I'm thankful to our partners in DNIT.

I thank also all the researchers, staff members, teachers and students who keep this University alive.

Abstract

In this work we review some important clustering methods, namely the K-Means and the soft K-Means in a probability framework, combined with optimization techniques. We give an overview of the main concepts and obstacles in unsupervised Learning with special emphasis on the overfitting challenge in high dimension which relates to the curse of dimensionality. We also present some recent techniques to overcome this phenomenon such as the ORCLUS algorithm. We present several illustrations of some of these clustering techniques (implemented in Python) and we also present some rigorous mathematical examples clarifying some obstacles for the clustering analysis in high dimension. We end this work with some applications to traffic counting real data from the Departamento Nacional de Infraestrutura e Transporte.

Keywords: Cluster Analysis, High Dimension, Unsupervised Learning, K-Means, ORCLUS

Contents

Introduction

Chapter 1

Supervised × Unsupervised Learning

1.1 Supervised Learning Overview

Say that we have a machine, A, generating random values in $\mathbb{R}^2$ and a line dividing the plane in two regions, H^+, the upper half of the plane, and H^-, the lower half. Now, let us suppose that every time the machine produces a value $\vec{X} \in H^+$, it is marked as red, and green when $\vec{X} \in H^-$. Next, we drop the line and send the data set generated to another machine, B. These two data sets are pictured below.

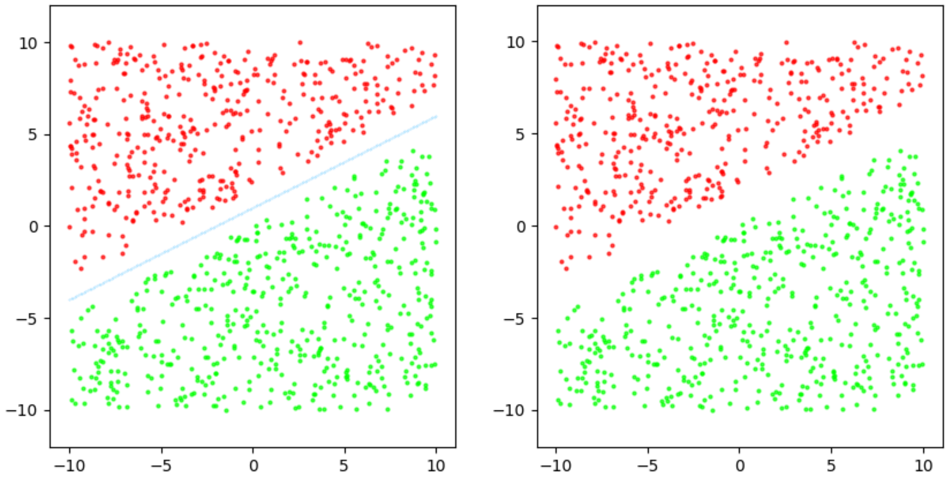


Figure 1.1: The points belonging to a cut of the plane by a linear function. On the left, the separator used by machine A, on the right the data how B receives.

If this machine B receives a newly generated random value, will it be able to decide if it is red

or green?
Humans are very likely to distinguish two spatial regions, check to which region a new data point is closest and mark it accordingly. However, this is not satisfactory, as one could argue that it is subjective and not reproducible. The aim of Machine Learning is to automate this kind of decision/labelling procedures, as objective and universal as possible, in such a way that computers are able to ascertain the right decisions or labellings to a good accuracy level.
There is a clear difference between human learning and machine learning: the former is much less complex, but more objective. Machines can perform rudimentary tasks, such as summations, multiplications or divisions, at a striking speed, but need to be told exactly how to do them. Therefore, in order to 'learn', a procedure to devise the special regions treated above, the programmer has to design the rules of label assignments. If a programmer working on machine B knew in advance that the initial data points were labelled according to a given line dividing the plane, that is, if they knew the specific model that generated the data, predicting new values would be a lot easier. The problem would size down to find a line that fits well the original division of colored points. What would be crucial is that the programmer has at hands a full set of data points already classified.
The a priori information set the model for the classification procedure: given a line $r = \{\alpha x + \beta, x \in \mathbb{R}\}$ with parameters α and $\beta \in \mathbb{R}$, the classification reduces to find a decision function Y that fits best with the observed empirical data. In this case the decision function is defined as

$$\begin{aligned} Y : \mathbb{R}^2 &\to \{red, green\} \\ x &\mapsto Y(x) \end{aligned}$$

$$Y(x) = \begin{cases} red & \text{if} \quad \alpha x + \beta < x \\ green & \text{if} \quad \alpha x + \beta \geqslant x \end{cases}$$

Y has parameters α and β. We will make use of *Machine Learning* methods to 'learn' what are the best estimates $\hat{\alpha}$ and $\hat{\beta}$ (for α and β respectively) that will allow us to predict newly generated data. This is different from finding the fittest estimates that best describes the observed data, a typical task of *Statistical Inference.* In fact, these two branchs go along together, as many algorithm like the *Expectation Maximization* discussed in chapter 3 perform a two step iteration alternating between an *inference* step and a *learning* step. We stress that *Machine Learning* deals with prediction of new data by classifying the state space points into distinct regions or by assigning them labels.
Back to our problem, the job of the programmer is to craft clever ways to automate the search for these parameters estimates. We should take into account what the machine "sees" and which operations can perform: it "sees" only data at memory and is able to perform few simple operations. A first rather naive method for making these estimates would be to randomly select α and β and scoring it according to a rule like counting the number of mistakes made. A more effective method would be the famous Perceptron Algorithm as described in [9].
These Learning algorithms normally start with an initial guess for the parameters estimators $\hat{\theta_0}$ and iteratively update it to values $\hat{\theta_1}, \hat{\theta_2}, \cdots \hat{\theta_n}$ until convergence (or a programmed stop).

It is important to guarantee convergence, but not always possible, and even when it does stop, rarely the optimal solution is achieved (as we will see in chapter 4 in the discussion of convergence towards local minima of the objective function of K-Means). For instance, consider if in our original problem data was labeled wrong, by mistake or deliberately, as in the Figure below.

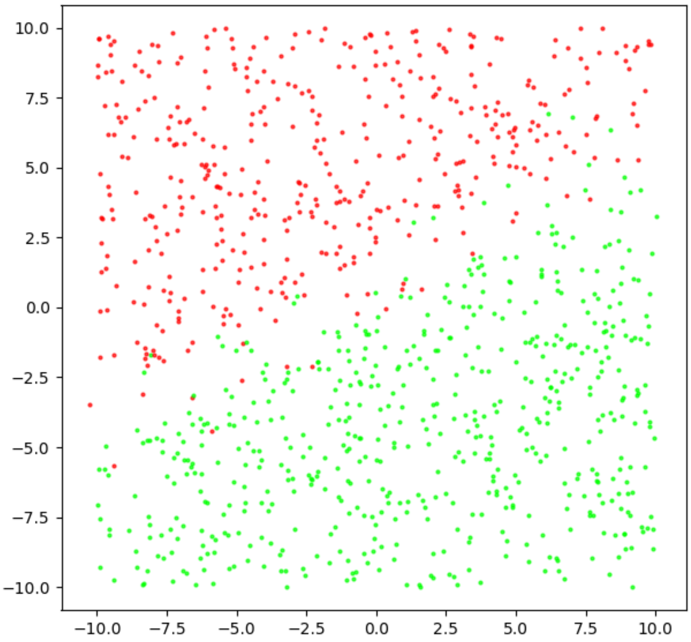


Figure 1.2: The picture contais the same data set but with some added noise around the separation region, such that it does not exist a perfect cut dividing the two regions.

Even though it is clear for a human analyst that there are two special regions, it will be impossible for the machine to arrive to a consistent conclusion through the Perceptron algorithm, as now convergence is not guaranteed. In other words, it is not possible any longer to divide the plane without committing assignment mistakes.

Another situation where perceptron fails is pictured in Figure 1.3. Fortunately, by performing a change of coordinates we can “linearize” the problem, i.e., to transform it in a simpler solvable problem.

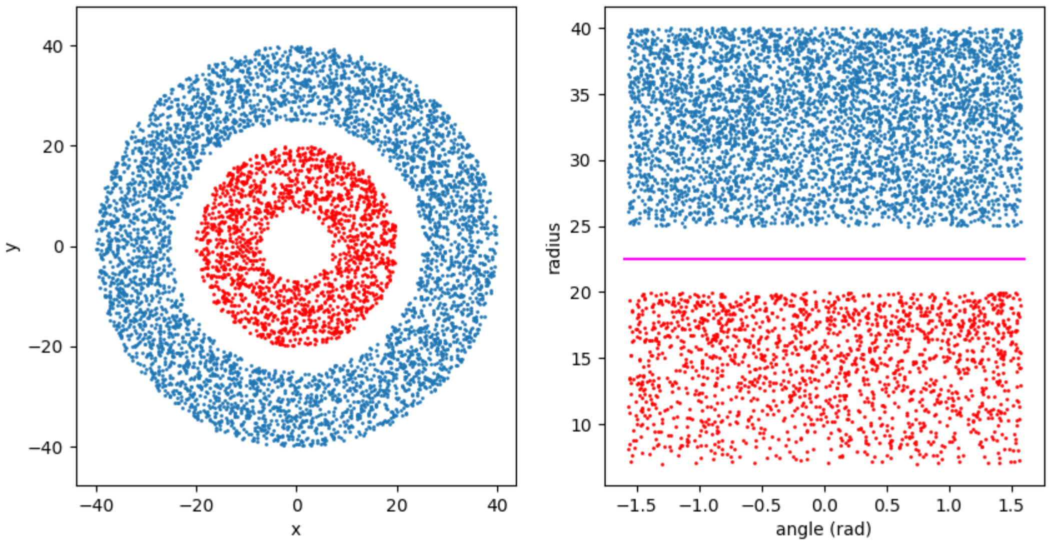


Figure 1.3: On the left we see points divided in two concentric regions. Clearly, the perceptron cannot divide the plane in two. However, a transformation to polar coordinates is enough to project the points in a space where two separable regions can be identified, as shown on the right.

Other methods project the data points into higher dimensional spaces, and, as we will see later in the Curse of Dimensionality in Chapter 5, the high dimensionality makes the distributed points more sparse and equidistributed, allowing to easily cut the space in two regions like H^+ and H^-.

All methods described here are deterministic in nature - a point is either red or green - meaning that the algorithm will not work properly if data is not originally shaped in deterministic terms.

Even after the coordinate transformation, there are points remaining outside the natural bunches expected (assuming this data is similar to the previous one). We should take care when considering generation of data that is deterministic against probabilistic in nature. Our model should account for fluctuations - for example, when scoring, we could simply count the mistaken values associated to a plane division by the line $r = \alpha x + \beta$ and select those values for α and β that minimize this mistakes scoring. One way we can learn from data and naturally account for probabilistic errors is through Linear Regression which is described in the next section.

1.2 Our first Supervised Learning tool: Linear Regression

Linear Regression is arguably the first Machine Learning tool a data scientist is introduced to. Suppose there is a quantity y that is related to another quantity x, and suppose we are given the empiric observations of pairs $x \times y$ values $\{(x_1, y_1), (x_2, y_2), \cdots\}$. Ideally, one would hope there is a law relating y's to x's, that could be expressed as a function $x \mapsto y(x)$, such that $y(x_i) = y_i$. Let's start assuming this relation is linear. Then

$$y(x) = \alpha x + \beta \tag{1.1}$$

for some two parameters that we aim to infer. In a perfect world, if the data had such a perfectly linear correlation, we would observe that for each $i = 1, 2, \cdots n$, $y_i = \alpha x_i + \beta$, but we observe instead

$$\begin{aligned} & y_i \neq \alpha x_i + \beta \\ \Rightarrow \quad & y_i = \alpha x_i + \beta + \epsilon_i \end{aligned}$$

where $\epsilon_i = y_i - \alpha x_i - \beta$ is the error observed in relation to the 'true' value $y_i = \alpha x_i + \beta$. We need to estimate α and β, whose estimators we denote $\hat{\alpha}$ and $\hat{\beta}$ respectively. We build an inference model assuming the hypothesis that the most probable values for α and β are those which make the error as small as possible. But how to measure the total error commited given two guesses for α and β?
We could define the total error as the length of the vector $\vec{\epsilon} = (\epsilon_1, \epsilon_2, \cdots \epsilon_n)$. Then $(\hat{\alpha}, \hat{\beta})$ solves a problem of the form:

$$\min_{\alpha,\beta} \ \| \ \vec{\epsilon} \ \|_{L^2} = \min_{\alpha,\beta} \ \| \ \vec{\epsilon} \ \|_{L^2}^2$$

Therefore, the estimators are calculated via

$$(\hat{\alpha}, \hat{\beta}) = \underset{\alpha,\beta}{\text{argmin}} \ \| \ \vec{\epsilon} \ \|_{L^2}^2 = \underset{\alpha,\beta}{\text{argmin}} \sum_{i=1}^{n} (\ y_i - \alpha x_i - \beta \ )^2$$

An example of a linear regression is presented in figure 1.2

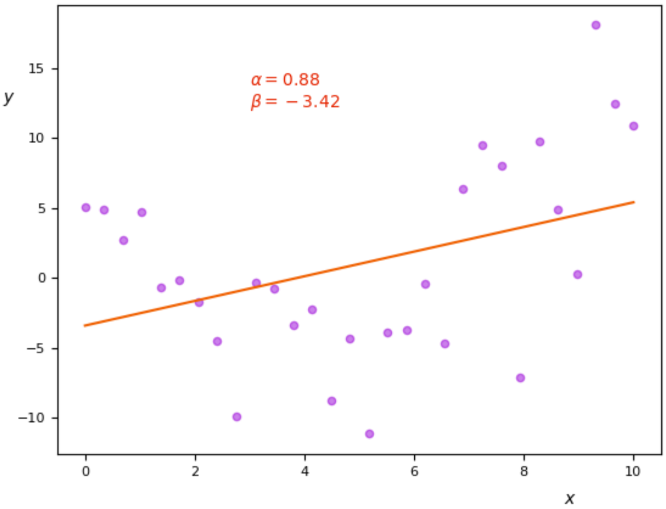


Figure 1.4: A random sample fitted by a linear curve. Despite the apparent correlation between x's and y's, the linear model does not sound a good fit for this sample.

Now assume we have a more complex phenomenon described by a quantity y depending on N features, described by the quantities $x^{(1)}, x^{(2)}, \cdots x^{(N)}$. We say y is a *dependent variable* which is measured as a '*response*' to the N *independent variables* x_i. We assume again the linearity hypothesis

$$y(x) = \omega_0 + \omega_1 x^{(1)} + \omega_2 x^{(2)} + \cdots + \omega_N x^{(N)} \tag{1.2}$$

The empirical observations are given by the set

$$D = \left\{ \left(x_1^{(1)}, x_1^{(2)}, \cdots x_1^{(N)}; y_1 \right),\ \left(x_2^{(1)}, x_2^{(2)}, \cdots x_2^{(N)}; y_2 \right), \cdots ,\left(x_n^{(1)}, x_n^{(2)}, \cdots x_n^{(N)}; y_n \right) \right\}$$

Again we define the error of each observation as

$$\epsilon_i = y_i - \omega_0 - \omega_1 x_i^{(1)} - \omega_2 x_i^{(2)} - \cdots - \omega_N x_i^{(N)}$$

and calculate the total error commited at each observation as the quantity $\| \vec{\epsilon} \|_{L^2}$. This model has more parameters, succinctly denoted $\omega = (\omega_1, \omega_2, \cdots \omega_N)$. We suppose once more that the coefficients ω_i are such to minimize the observed total error. So the estimate for ω is the solution of the problem

$$\min_{\omega} \| \vec{\epsilon} \|_{L^2}^2$$

whose solution is

$$\begin{aligned} (\hat{\omega_1}, \hat{\omega_2}, \cdots \hat{\omega_N}) &= \underset{\hat{\omega}_1, \hat{\omega}_2, \cdots \hat{\omega}_N}{\operatorname{argmin}} \quad \| \vec{\epsilon} \|_{L^2}^2 = \\ &= \underset{\hat{\omega}_1, \hat{\omega}_2, \cdots \hat{\omega}_N}{\operatorname{argmin}} \sum_{i=1}^{n} \Big(\ y_i - \omega_0 - \omega_1 x_i^{(1)} - \omega_2 x_i^{(2)} - \cdots - \omega_N x_i^{(N)} \ \Big)^2 \end{aligned}$$

The quantity

$$E(\omega) = \sum_{i=1}^{n} \Big(\ y_i - \omega_0 - \omega_1 x_i^{(1)} - \omega_2 x_i^{(2)} - \cdots - \omega_N x_i^{(N)} \ \Big)^2$$

is said to be the *total squared error* which we seek to minimize.
The observed values $\{x_i^{(j)}\}_{i=1,2,\cdots n}^{j=1,2,\cdots N}$ and their *responses* $\{y_i\}$ can be represented respectively by the following matrix and vector

$$X = \left[\begin{array}{c|c} 1 & \vdots \\ \vdots & x_i^{(j)} \\ 1 & \vdots \end{array} \right] \quad , \quad \mathbf{y} = (y_1, y_2, \cdots y_n)$$

for $i = 0, 1, 2, \cdots n$ and $j = 1, 2, \cdots N$. The first columns full of 1's is needed to multiply the parameter ω_0. In this notation E and the values for $\hat{\omega}_j$ thus read

$$\begin{gathered} E(\omega) = \|\mathbf{y} - X\omega\|_{L^2}^2 \\ \\ (\hat{\omega}_0, \hat{\omega}_1, \cdots \hat{\omega}_N) = \underset{\hat{\omega}_1, \hat{\omega}_2, \cdots, \hat{\omega}_N}{\operatorname{argmin}} \|\mathbf{y} - X\omega\|_{L^2}^2 \end{gathered} \tag{1.3}$$

1.3 The kernel trick

We will calculate explicitly the solution to problem 1.3 further in the next section. Now we go back to the starting point of last section: we are given the set $D = \{(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\}$ and we want to model it so that ideally y depends on the observed variables through the relation $y = y(x)$. If we find such a relation we would be able to *predict* how y would behave in relation to newly observed x's. The linear model in general gives a very poor estimate of the data, because it assumes a linear correlation between x and y. It does so because there are few parameters to account for shapes peculiarities. Think of the following: assuming $y = y(x)$ is differentiable, then if we expand it in Taylor series we find

$$y(x) = y(0) + y'(0)x + \frac{1}{2}y''(0)x^2 + \frac{1}{3!}y'''(0)x^3 + \cdots \frac{1}{n!}y^{(n)}(0)x^n + r(x^{n+1})$$

If the derivatives are all bounded above a certain n_0, we would expect that at some point, say at the m^{th} derivative, $m!$ is big enough to make the derivative $f^{(m)}(0)$ irrelevant (the same applying for $l \geq m$), thus we discover that the function y could be truncated at m. So we have m coefficients to discover. Whatever model we choose to predict $y = y(x)$ given the observed pairs (x_i, y_i), it is very likely to possess more than two parameters, and there is a chance that no more than m parameters will be needed.

The linear model 1.2 describes the function $y = y(x)$ with $N + 1$ parameters ω_j but assumes y is a function of N features $x^{(1)}, x^{(2)}, \cdots x^{(N)}$, but notice that nothing is said about whether these features are independent or not. So we use a clever trick in order to insert more parameters to the model describing y: we fix a N and construct artificially the tuple of observations

$$\left(x_i^{(0)}, x_i^{(1)}, x_i^{(2)}, \cdots x_i^{(N)}; y_i\right) := \left(1, x_i, (x_i)^2, \cdots (x_i)^N; y_i\right)$$

for $i = 1, 2, \cdots n$. Notice that we considered the 1's that appeared in the matrix X a new feature $x^{(0)}$ thus making the former be rewritten as

$$X = \left[\ x_i^{(j)}\ \right] = \left[\ (x_i)^j\ \right]$$

for $i = 1, 2, \cdots n$ and $j = 0, 1, 2, \cdots N$. What we have done was to lift the feature represented by the variable $x \equiv x^{(1)}$ to a higher dimensional space of $N + 1$ features $x^{(0)}, x^{(1)}, x^{(2)}, \cdots x^{(N)}$, through the function

$$\phi(x^{(1)}) = \left(\phi_0(x^{(1)}), \phi_1(x^{(1)}), \phi_2(x^{(1)}), \cdots, \phi_N(x^{(1)})\right)$$

with

$$\phi_j\left(x^{(1)}\right) = \left(x^{(1)}\right)^j$$

Each ϕ_j is said to be a *kernel function.* There are more *kernels* that could be used to lift the feature $x^{(1)}$, like $\phi_{\sigma,\mu}(x) \propto \exp((x-\mu)^2/2\sigma^2)$, so in general we expect to generate the matrix

$$\Phi = \left[\ x_i^{(j)}\ \right] = \left[\ \phi_j(x_i^{(1)})\ \right] \tag{1.4}$$

instead of X, thus producing the total squared error

$$E(\omega) = \|\mathbf{y} - \Phi\omega\|^2_{L^2}$$

that we seek to minimize in order to find

$$(\hat{\omega}_0, \hat{\omega}_1, \cdots \hat{\omega}_N) = \underset{\omega_1, \omega_2, \cdots, \omega_N}{\operatorname{argmin}} E(\omega)$$

This clever trick of lifting the data set to a higher dimensional feature space making use of kernels ϕ_j is called the *kernel trick*. It works because in higher dimensions the transformed data points in general become distributed in a sparse manner, making it easy to find hyperplanes which describes well the data - it is like there was more space to accomodate them optimaly. On Chapter 5 we will discuss in depth this phenomenon called the *curse of dimensionality* (in this case it is a bless!). Further details and calculations on regression linear techniques can be seem at [8].

1.4 Overfitting

Let's test the *kernel trick* with the linear regression using the kernels $\phi_j(x) = x^j$. This combination gives the technique known as *polynomial regression* or *polynomial fit*. Find below some regressions performed with increasing numbers of parameters, namely $N = 0, 1, 3, 9$. One could think that the more parameters we have at disposal the better we could fit the data, and in fact yes, as verified for $N = 9$. However, once we are told that the points were sampled from a sinus with noise we could not say the linear regression with $N = 9$ is a good model choice.

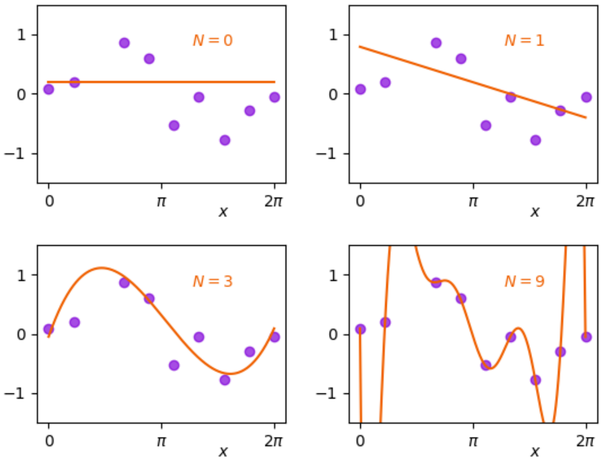


Figure 1.5: Four regressions with increasing number N of parameters, as indicated. The original curve which generated these points was a *sinus* with some added noise. Notice that $N = 9$ gives a very bad prediction of $y = y(x)$ while $N = 3$ is very accurate in predicting a *sinus*.

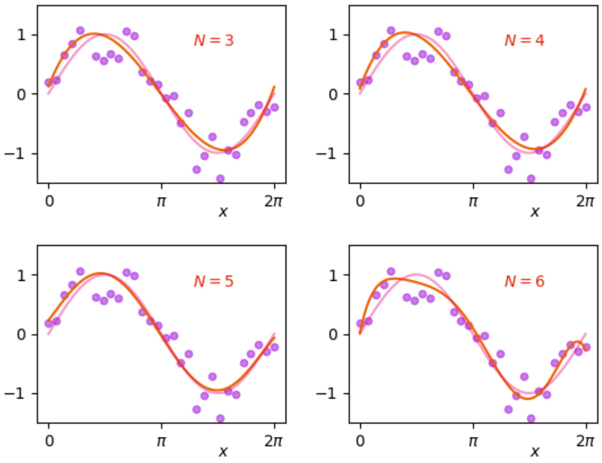


Figure 1.6: We investigate more carefully if $N > 3$ performs better. Notice that the best fit seems to be $N = 5$ while for $N = 6$ the regression has performs badly. The original curve is displayed in pink

The problem is that when there are too many parameters in comparison to the number of samples and the underlying complexity of their distribution, the model begins to accomodate all the points in its fit, giving unrealistic solutions that incorporates noise as they were genuinely inherent of the sample generation process. These solutions fit very well the data points but lack predictive power, as we evidentiate in the botton of the last column of Figure 1.5 or in the first column Figure 1.7 . This phenomenon of fitting the errors due to presence of many parameters is named *overfitting*. In Figure 1.6 we investigate for what values for N the regression becomes overtfitted.

Now, there is a way to take benefit from having many parameters at disposal while significantly reducing the *overfitting*. We do it by regularizating the *squared error function* by adding a new term which will penalize coefficients ω_j associated to a high degree of overfitting. Typically the overfitting solutions are associated to very large polynomial coefficients ω_j, so we can simply add a term proportional to $\|\omega\|_{L^2}^2$, which leads to the problem

$$\min_{\omega} \left\{ \|\mathbf{y} - \Phi\omega\|_{L^2}^2 + \frac{\lambda}{2}\|\omega\|_{L^2}^2 \right\}$$

The aforementioned problem constitutes the so-called *ridge linear regression* with one penalty term λ. We test this technique on the second and third columns of figure 1.7 for crescent values of N and two values of λ, against the pure linear regression on the frst column. It was tested for many other values of λ and many N's, and it was concluded that until $N = 29$ to small values of λ like 0.1, 10^{-3} and even 10^{-15} it corresponds perfect fit like those pictured for $N = 25$ (notice that the original curve which generated the samples is drawn in the backgrond).

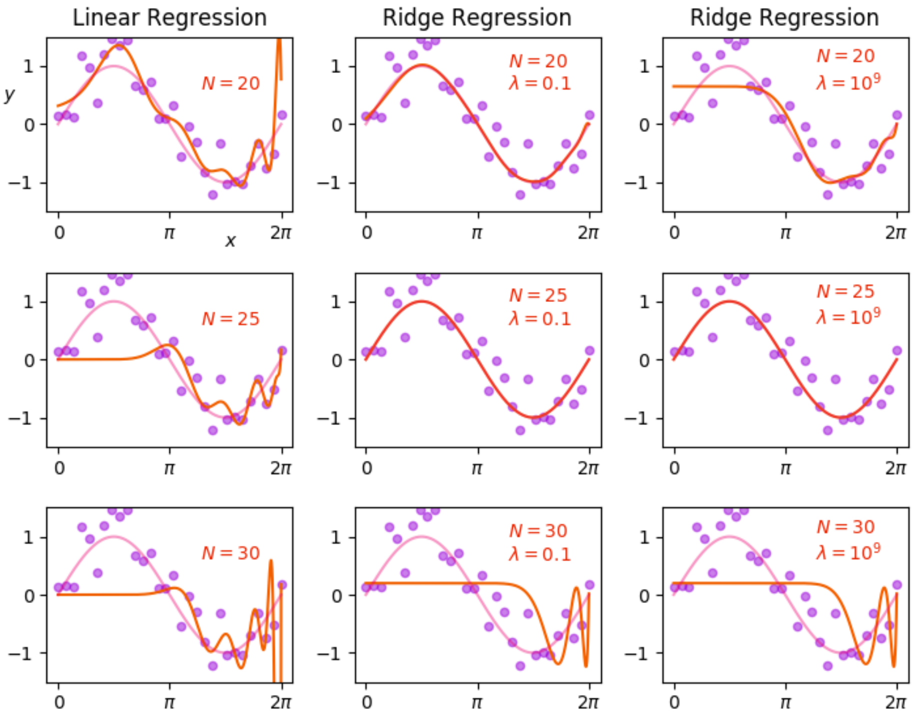


Figure 1.7: Ridge rgression is tested against normal linear regression. On the second column we test it with $\lambda = 0.1$, on the third, $\lambda = 10^9$.

In this work we will pay close attention to the overfitting of data points. Our goal is to study *Cluster Analysis* in high dimensions. In this regime the data becomes naturally sparse - making it very difficult to measure distance between two points - due to the already mentioned phenomenon called *curse of dimensionality.* Hence we are naturally led to to Machine Learning methods that either possess many parameters or have low predictive power due to this inherent noise on the data. We proceed further developing a solution to the problem

$$E(\omega) = \|\mathbf{y} - X\omega\|^2_{L^2}$$

$$(\hat{\omega}_0, \hat{\omega}_1, \cdots \hat{\omega}_N) = \underset{\hat{\omega}_1, \hat{\omega}_2, \cdots, \hat{\omega}_N}{\operatorname{argmin}} \|\mathbf{y} - X\omega\|^2_{L^2}$$

after placing the *least squares* principle in a probabilistic framework.

1.5 A probabilistic context to Linear Regression

In Linear Regression, for each x, there is a corresponding value $y(x)$ that could only be measured in a total absence of pertubations to the measurement process. As this is impossible, we add an error variable to y measurement on x. In this way, our measurement over x becomes a random variable $Y_x = y(x) + \epsilon_x$, where ϵ_x is a random variable following a gaussian distribution $\rho(s) = \frac{\mathrm{e}^{-s^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}}$. The variable ϵ_x can be thought as a noise added to the actual value for $y = f(x)$ by any random (or unknown) process. We assume independency between Y's measured at different x's, *i.e.*, the added noises are uncorrelated.

These are strong assumptions, but making them has the big payoff of simplicity. A good practice in computing applications is to first start with the simpler approaches, and gradually, if needed, move to more complicated ones. Moreover, we should point out that it is very common to observe statistical fluctuations following a gaussian pattern, in agreement with the Central Limit Theorem, which justifies our first hypothesis. The independency assumption is also well justified in systems where the measurement of y at point x does not depend on, or does not have memory of, what was measured in a previous point x'.

We assume $y = y(x)$ is a polinomial, so we can write Y as $Y_x = \sum_{k=0}^{N} \omega_k x^k + \epsilon_x$. Now, Y_x has a gaussian distribution centered at $y(x)$, and the parameters to be set here are the ω coefficients and σ. Thus the probability density of Y_x is

$$p(Y_x = y \mid \omega, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-(y-y(x))^2/2\sigma^2}$$

Where $\omega = (\omega_1, \omega_2, \cdots \omega_N)$, $y(x) = \sum_{k=0}^{N} \omega_k x^k$, N is the degree of the polinomial we chose in advance. Notice that N is not a probabilistic parameter - it cannot be inferred by statistical methods as those described below. N should be fixed by hand, and can be chosen so to minimize the *bias-variance trade off.* We refer to [33] for further detail, but for us N is just a fixed parameter for our model.

Give the observed data $D = \{(x_1, y_1), (x_2, y_2), \cdots\}$, we have the probability of occuring D:

$$p(D \mid \omega, \sigma) = p(Y_{x_1} = y_1, Y_{x_2} = y_2, \cdots Y_{x_N} = y_N \mid \omega, \sigma) = \prod_{k=0}^{N} p(Y_{x_k} = y_k \mid \omega, \sigma)$$

Where $p(Y_{x_k} = y_k \mid \omega, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-(y_k - y(x_k))^2/2\sigma^2}$ and $y(x_k) = \sum_{i=0}^{N} \omega_i x_k^i$.

Now, we need to define estimators $\hat{\omega}$ and $\hat{\sigma}$ for their respective parameters. We are going to look for the values of σ and ω that maximize the probability of occuring D, *i.e.*, those parameters that characterize a universe where D is the least rare possibility, and where we assume we live in. So we define the Maximum Likehood Estimator (MLE) for the parameters as the values $\hat{\sigma}_{MLE}$ and $\hat{\omega}_{MLE}$ maximizing the likehood funtion

$$\mathcal{L}^{(D)}(\omega, \sigma) = p(D \mid \omega, \sigma)$$

or equivalently, those maximizing the logarithmic of $\mathcal{L}^{(D)}$:

$$l^{(D)}(\omega,\sigma) = \ln\left(\prod_{k=0}^{N} p(Y_{x_k} = y_k \mid \omega,\sigma)\right) = \sum_{k=0}^{N} -\frac{(y_k - y(x_k))^2}{2\sigma^2} - \ln\sqrt{2\pi} - \ln\sigma$$

Thus giving

$$l^{(P)}(\omega,\sigma) = -(N+1)\ln\sqrt{2\pi} - (N+1)\ln\sigma - \frac{1}{2\sigma^2}\sum_{k=0}^{N}\epsilon_{x_k}^2$$

So, we define our estimators as

$$\begin{aligned} \hat{\omega}_{MLE} &= \underset{\omega}{\text{argmax}}\ \ l^{(D)}(\omega,\sigma) \\ \hat{\sigma}_{MLE} &= \underset{\sigma}{\text{argmax}}\ \ l^{(D)}(\omega,\sigma) \end{aligned} \tag{1.5}$$

Now,

$$\begin{aligned} \underset{\omega}{\text{argmax}}\ \ l^{(D)}(\omega,\sigma) &= \underset{\omega}{\text{argmax}}\left\{-\frac{1}{2\sigma^2}\sum_{k=0}^{N}\epsilon_{x_k}^2\right\} = \underset{\omega}{\text{argmin}}\left\{\sum_{k=0}^{N}\epsilon_{x_k}^2\right\} \\ \underset{\sigma}{\text{argmax}}\ \ l^{(D)}(\omega,\sigma) &= \underset{\sigma}{\text{argmax}}\left\{-(N+1)\ln\sigma - \frac{1}{2\sigma^2}\sum_{k=0}^{N}\epsilon_{x_k}^2\right\} = \\ &= \underset{\sigma}{\text{argmax}}\left\{(N+1)\ln\sigma + \frac{1}{2\sigma^2}\sum_{k=0}^{N}\epsilon_{x_k}^2\right\} \end{aligned}$$

Finally, we derive the expressions inside the braces and set them equal to zero and solve for our MLE estimators. We begin with the σ:

$$0 = \frac{\partial}{\partial\sigma}\left\{(N+1)\ln\sigma + \frac{1}{2\sigma^2}\sum_{k=0}^{N}\epsilon_{x_k}^2\right\} = \frac{N+1}{\sigma} - \frac{1}{\sigma^3}\sum_{k=0}^{N}\epsilon_{x_k}^2$$

$$\therefore\quad \sigma^2 = \frac{1}{N+1}\sum_{k=0}^{N}\epsilon_{x_k}^2$$

We notice that

$$\sigma^2 = \left(\frac{N}{N+1}\right)\frac{1}{N}\sum_{k=0}^{N}(y_k - y(x_k))^2 \approx \left(\frac{N}{N+1}\right)s^2$$

where s^2 is the empirical variance.
Let's solve now for each $\omega_i \in \boldsymbol{\omega}$:

$$0 = \frac{\partial}{\partial \omega_i} \left\{ \sum_{k=0}^{N} \left(y_k - \sum_{j=0}^{N} \omega_j {x_k}^j \right)^2 \right\} = \sum_{k=0}^{N} \left\{ -2 \left(y_k - \sum_{j=0}^{N} \omega_j {x_k}^j \right) \left(\sum_{j=0}^{N} \frac{\partial \omega_j}{\partial \omega_i} {x_k}^j \right) \right\}$$

$$0 = \sum_{k=0}^{N} \left(y_k - \sum_{j=0}^{N} \omega_j {x_k}^j \right) {x_k}^i$$

$$\therefore \quad \sum_{k=0}^{N} {x_k}^i \left(\sum_{j=0}^{N} {x_k}^j \omega_j \right) = \sum_{k=0}^{N} {x_k}^i y_k \quad \forall i = 0, 1, \cdots N$$

If we define the matrix X by $X_{ij} = {x_i}^j$ (x_i's are the positions of the measurements, which are exponentiated by each j), then the above relation reads:

$$(X^T X) \hat{\omega}_{MLE} = X^T \mathbf{y}$$

where $\mathbf{y} = (y_0, y_1, \cdots y_N)$ is the vector of observations of quantity y. Now, the measure of the set $\{(x_0, x_1, \cdots x_N) \mid \det(X^T X) = 0\}$ is zero, meaning that almost surely ("a.s.") one can invert the matrix in question, thus giving us:

$$\hat{\omega}_{MLE} = (X^T X)^{-1} X^T \mathbf{y} \quad \text{a.s.} \tag{1.6}$$

This is the classical problem of the least squares in Linear Algebra. In turns out that we transformed a probabilistic problem into a pseudo-inverse calculation of a matrix X. Computationally, there is no much effort in calculating it as long as the dimensionality, dictated by N, is not too high. It is worth noting that all the calculations above remain the same if we use different kernels than $\phi_j(x) = x^j$, but only replacing X by Φ (defined in 1.4).

Is this really Machine Learning? Or is it simply another Statistical Inference method? It depends on how we define "learning" and to what extent we look for a division between these two subjects. Despite a discussion on their difference goes beyond the scope of this thesis, stress that Machine Learning methods frequently involve a loop of calculations of a sequence of estimators, until their convergence - and this process is not something that could be easily done with a calculator, instead this is typically a computer application. Hence, we can think of Machine Learning as a set of tools and applied statistics knowledge designed especially for computers and machines alike. Also, Machine Learning is deeply concerned in predicting patterns assocated to newly generated pieces of data, based on some training set.

Why do we say the Maximum Likehood Estimator method is "supervised"? Because we were told how the x_i's related to their respective y_i's. This is already a lot of information. Supervised Learning means that we know something about the inner structure of the different pieces of data. Suppose that instead of the pairs of D, we were only given the two sets $\mathcal{X} = \{x_0, x_1, \cdots x_N\}$ and $\mathcal{Y} = \{y_0, y_1, \cdots y_N\}$ but no one told us how to associate them in pairs (as the labellings suggest...). Now, not only we have to predict what shape the fitting curve should have, but we should infer the most probable configuration of pairs (x_i, y_j) among $\mathcal{X} \times \mathcal{Y}$.

So, Unsupervised Learning is concerned with finding structure in apparently unstructured data by relying mainly on the data itself, or in other words, is concerned with finding any kind of pattern that can be understood as the *a priori* structure existing in the data. The main difficulty of these methods is to validate them properly, as we do not have classified in advance data to compare with, and actually, in most situations we do not even have an intuition of the possible classification. Therefore, they heavily rely on strong assumptions about the shape of the data. For example, we will see in the next chapter that the K-Means algorithm assumes implicitly that the distribution of data points in $\mathbb{R}^d$ is approximately in spherical blobs, all with the same radius.

Classification of the data can be an expensive process - for example, imagine how a Telecom company could classify each one of its millions of users. Because of it, Unsupervised Learning methods are becoming increasingly important to find patterns in data sets which is the first step for most data analysis.

In the next chapter, we discuss the K-Means algorithm, one of the most important tool for Unsupervised Learning for its simplicity and speed of convergence.

Chapter 2

Unsupervised Learning

2.1 What is "learning"?

In the previous chapter we introduced ideas and language of *Supervised Learning* methods to prepare the ground for the topic of this thesis, namely, *Unsupervised Learning.* The typical problem of *Unsupervised Learning* is classification. There are: (i) a state space of all accessible data points; (ii) labels to classify these points, *i.e.*, a proper division of this space into subspaces one for each label; and (iii) the observed empirical data points to be classified.
For example, consider a data set in a spreadsheet format (like a '*.csv*' or a '*.xml*') where each row represents a costumer of a certain company, and each column is a feature of this costumer, like "age","gender", "civil status" and "credit status". Suppose that we are working with only "age", "gender" and "credit status". The feature "age" can be treated as a natural number variable; "gender" as a categorical variable with values of "M" (male) or "F"(female); "credit status" as a real number that is the total debt or credit.
So, in this example, our state space is $\mathcal{X} = \mathbb{N} \times \{``M", ``F"\} \times \mathbb{R}$. Now, if we want to classify the costumers into creditors or debtors, according to whether they have positive value in their accounts or not, we look at the "credit status" variable and check if it is positive or negative. This is equivalent to dividing $\mathcal{X}$ in two disjoint regions, $\mathcal{X}^+ = \{(n, g, d) \in \mathcal{X} \mid d \geqslant 0\}$ and $\mathcal{X}^- = \{(n, g, d) \in \mathcal{X} \mid d < 0\}$.
So, given a table of data points where each row represents a costumer status, one can classify them as a debtor or as a creditor. Denoting the observed data set as $D = \{x_1, x_2, \cdots x_N\} \subset \mathcal{X}$, where $x_i = (n_i, g_i, d_i)$ is the i^{th} row of the spreadsheet, these data points can be clustered in two groups: $C^+ = \{x_i \in D \mid d_i \geqslant 0\}$ and $C^- = \{x_i \in D \mid d_i < 0\}$
We can equivalently define a classification function $Y : \mathcal{X} \to \{0, 1\}$, where 0 means 'debtor' and 1 means 'creditor', as

$$Y(n, g, d) = \begin{cases} 1 & if \quad d \geqslant 0 \\ 0 & if \quad d < 0 \end{cases}$$

So that the two regions of $\mathcal{X}$ are the inverse images of Y:

$$\mathcal{X}^+ = Y^{-1}(1) \text{ and } \mathcal{X}^- = Y^{-1}(0)$$

And the clusters are simply:

$$C^{+} = Y^{-1}(1) \cap D \ \text{ and } \ C^{-} = Y^{-1}(0) \cap D$$

As the reader might have noticed, this is a trivial case. We defined $Y = 1$ for costumers with positive credit status, and $Y = 0$ for those with debts, negative credit status. There is nothing to be learnt here.
The process of learning comes into place when we do not know for certain the criteria for data classification, or in other words, we do not know the Y function with precision. Back to our example, we could consider the following criteria: (i) the costumer that always has positive credit status; (ii) the costumer who has a negative credit status but it looks like a temporary situation, for example the costumer forgot to pay the bill; (iii) the costumer always has negative credit status.
We could divide clients into N categories corresponding to N segments $\{[a_i, b_i)\}_{i=0,1,\cdots N-1}$ of the variable $d \in \mathbb{R}$. Therefore, the new classification function would be

$$Y(n, g, d) = \ \ i \ , \quad if \quad d \in [a_i, b_i)$$

Here, 'learning' consists in finding the best fit classification function Y based on the data set itself. In the example, this is equivalent to find the suprema and infima $\{a_i, b_i\}_{i=0,1,\cdots N}$ that best describe the segment groups of costumers.
In Supervised Learning methods, we know in advance the classification for a (small) part of the costumers (for example, in credit companies, the data scientists rely on previous experiences recorded in old databases, back then the selection of good clients, worthy of granting credit, was made by hand by the human analysts), we have some good examples we can trust, and we use them to infer somehow the classification of the remainng data. We seek to *predict* the class the data belong to, but the premise is that we know what the classes are!
For Unsupervised problems, on the opposite, there are no past examples, no already classified data we can rely on. The aim is to *recognize patterns* from scratch! Or at least from weaker assumptions. For this example, one could look for dense regions on the line, and set boundaries to these regions, using some criteria that maximize the density for each (so-called) *cluster*.
In the end what matters are the clusters themselves rather than the classification function. Unsupervised Learning deals mainly with Cluster Analysis and its implementations, many of which are extensions or generalizations of the object of study of the present thesis, the K-Means. Therefore, as we will see, many of these methods will translate into a non-linear optimization problem.
Given some data points $D = \{x_1, x_2, \cdots, x_N\}$, where each x_i is an element of the state space $\mathcal{X}$, we can preliminiraily define a Clustering of D as a partition $\{C_1, C_2, \cdots, C_M\}$ such that $\dot{\bigcup}_{i=1}^{M} C_i = D$, in which by 'cluster' we mean each of the subsets C_i.
As in our definition the union is disjoint, one can classify data according to its belonging to the i^{th} cluster C_i. This definition actually stands for a *Hard Clustering* of D. We will see that a more general definition is to define the rate of belonging of each point to each one of k classes, that we will call a *Soft Clustering*. This object will be defined in Chapter 3.

In the next section, to make things concrete we analyse a first tool in Cluster Analysis, the K-Means algorithm.

2.2 The K-Means Algorithm

The first unsupervised learning tool we present is the K-Means algorithm. The K-Means is a two-step iterative algorithm. The first step consists in dividing the data in clusters according to their closeness to reference vectors, called centroids, representing each cluster. In the second step we update the centroids to new cluster representatives, based on the previous classification.

Let us consider the set of data points $D = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_N\}$ and assume there exists a distance function, or metric, $(\mathbf{x}, \mathbf{y}) \mapsto d(\mathbf{x}, \mathbf{y})$ defined over $\mathbb{R}^d$ indicating how 'close' two data points are from each other. The algorithm is initialized with k-tuple of centroids denoted by

$$\mathbf{M}^{(0)} = (\mathbf{m}_1^{(0)}, \mathbf{m}_2^{(0)}, \cdots \mathbf{m}_k^{(0)}) \in \mathbb{R}^d \times \mathbb{R}^d \times \cdots \times \mathbb{R}^d$$

The first step consists in assigning the nearest centroid to each data point $\mathbf{x}_i \in D$ using the classification function

$$Y \colon \mathbb{R}^d \to \{1, 2, \cdots k\}$$

$$Y(\mathbf{x}) = \underset{j \in \{1,2,\cdots k\}}{\operatorname{argmin}} \; d(\mathbf{x}, \mathbf{m}_j^{(0)})$$

Remark. *Notice that the above definition is unclear when* $\mathbf{x}$ *is equidistant to two centroids as they both minimize the function* $d(\mathbf{x}, \mathbf{m}_l)$*. So we redefine the* argmin *function as*

$$\underset{j \in \{1,2,\cdots k\}}{\operatorname{argmin}} \; d(\mathbf{x}, \mathbf{m}_j) = \min\{j \;\mid\; d(\mathbf{x}, \mathbf{m}_j) \leq d(\mathbf{x}, \mathbf{m}_l) \;\forall\; l\}$$

We acknowledge that this choice is arbitrary, however we notice that the data points that are equidistant from two centroids belong to the border of the so-called Voronoi sets, which have zero measure in $\mathbb{R}^{\mathrm{d}}$*. Therefore data points satisfying this property are quite rare.*

We use the definition above to define the k clusters

$$C_i = Y^{-1}(i) = \{\mathbf{x} \in D \mid \underset{j \in \{1,2,\cdots k\}}{\operatorname{argmin}} \; d(\mathbf{x}, \mathbf{m}_j^{(0)}) = i\}$$

for i=1,2,$\cdots k$

These are the sets of points that lie closest to the i^{th} centroid.

In the second step we average over each cluster to construct the next k-tuple $\mathbf{M}^{(1)}$ by updating the centroids to

$$\mathbf{m}_i^{(1)} = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x} \quad \text{for} \quad i = 1, 2, \cdots k \tag{2.1}$$

This brings us back to the first step, where we re-evaluate for every $\mathbf{x} \in D$ its closest centroid $\mathbf{m}_i^{(1)}$ of $\mathbf{M}^{(1)}$ and assign for it the label i. Then we continue the iteration similarly to produce $\mathbf{M}^{(2)}, \mathbf{M}^{(3)}, \cdots$. We will prove in the next section, 2.2.1, that for some iteration $n \in \mathbb{N}$, $\mathbf{M}^{(n+1)} = \mathbf{M}^{(n)}$, so the centroids reach a fixed position and further iterations do not change the clusters. The basic algorithm (n^{th}) iteration is summarized as

Assign: For every $\mathbf{x} \in D$ and the current centroids $\mathbf{m}_i^{(n)}$ for $i = 1, 2, \cdots k$, calculate the distances $d(\mathbf{x}, \mathbf{m}_i^{(n)})$ and group the points into k clusters $C_i^{(n)}$ according to the assignment

$$i = \underset{j \in \{1,2,\cdots k\}}{\operatorname{argmin}} \; d(\mathbf{x}, \mathbf{m}_j) \tag{2.2}$$

Update: Update the centroids from $\mathbf{m}_i^{(n)}$ to:

$$\mathbf{m}_i^{(n+1)} = \frac{1}{|C_i^{(n)}|} \sum_{\mathbf{x} \in C_i^{(n)}} \mathbf{x} \quad \text{for} \quad i = 1, 2, \cdots k \tag{2.3}$$

Now, we rewrite expression 2.3 in a less elusive form by defining below the so called *responsibility* function associated to each point $\mathbf{x} \in D$. The *responsibility function* gives a more friendly notation for the assignment step of cluster points at the n^{th} iteration, and allows to write the expression for $\mathbf{m}_i^{(n+1)}$ in a cleaner way so to suggest a generalization to be discussed later in Section 2.3. The *responsibility* function at n^{th} iteration is

$$r_{\mathbf{x}}^{(n)}(i) = \delta(i, Y(\mathbf{x})) = \begin{cases} 1, & \text{if} \quad \underset{j \in \{1,2,\cdots k\}}{\operatorname{argmin}} \; d(\mathbf{x}, \mathbf{m}_j^{(n)}) = i \\ 0, & \text{if} \quad \underset{j \in \{1,2,\cdots k\}}{\operatorname{argmin}} \; d(\mathbf{x}, \mathbf{m}_j^{(n)}) \neq i \end{cases} \tag{2.4}$$

Notice that at each iteration, $Y(\mathbf{x})$ is updated taking into account the values of the centroids $\mathbf{m}_i^{(n)}$. Therefore, r is a function that depends on n as indicated by the "(n)" superscript. The responsibility $r = r_{\mathbf{x}}^{(n)}(i)$ is an indicator function and gives 1 when $\mathbf{x}$'s nearest centroid has index i, and 0 otherwise. Therefore $\mathbf{x} \in C_i$ if and only if $r_{\mathbf{x}}^{(n)}(i) = 1$ and for every point $\mathbf{x} \notin C_i$, $r_{\mathbf{x}}^{(n)}(i) = 0$, so we infer the relations

$$\sum_{\mathbf{x} \in D} r_{\mathbf{x}}^{(n)}(i) = \sum_{\mathbf{x} \in C_i} 1 = |C_i|$$

$$\sum_{\mathbf{x} \in D} r_{\mathbf{x}}^{(n)}(i) \; \mathbf{x} = \sum_{\mathbf{x} \in C_i} \mathbf{x}$$

We can re-express the ***Update*** step that produces the new centroids in 2.3 as

Update: Update the centroids from $\mathbf{m}_i^{(n)}$ to:

$$\mathbf{m}_i^{(n+1)} = \frac{\sum_{\mathbf{x} \in D} r_{\mathbf{x}}^{(n)}(i) \; \mathbf{x}}{\sum_{\mathbf{x} \in D} r_{\mathbf{x}}^{(n)}(i)} \quad \text{for} \quad i = 1, 2, \cdots k \tag{2.5}$$

As the responsibility function takes only 0 or 1 values, the K-means method we described is said to be a '*hard clustering*'. Later in the thesis, we will generalize the responsibility function and the centroid expressions to the *soft clustering* case. K-means is an algorithm that produces a sequence of tuples $\mathbf{M}^{(n)}$ up to a fixed point satisfying $\mathbf{M}^{(n+1)} = \mathbf{M}^{(n)}$. In the next section we describe convergence. The following pseudocode summarizes the algorithm.

Algorithm 2.2.1: K-MEANS(k, D)

```
for $i \leftarrow 1$ to $k$
  do $\mathbf{m}_i^{(0)} \leftarrow$ random value

$\mathbf{M}^{(0)} \leftarrow (\mathbf{m}_1^{(0)}, \cdots \mathbf{m}_k^{(0)})$

$\mathbf{M}^{(1)} \leftarrow (\mathbf{0}, \cdots \mathbf{0})$

$(l_1, l_2, \cdots l_N) \leftarrow (0, 0, \cdots 0)$

while $\mathbf{M}^{(1)} \neq \mathbf{M}^{(0)}$
  do
    comment: calculate each $Y(\mathbf{x}_i)$
    for each $\mathbf{x}_i \in D$
      do
        $s \leftarrow d(\mathbf{x}_i, \mathbf{m}_1^{(0)})$
        for $j \leftarrow 1$ to $k$
          do
            if $d(\mathbf{x}_i, \mathbf{m}_k^{(0)}) < s$
              then
                $s \leftarrow d(\mathbf{x}_i, \mathbf{m}_k^{(0)})$
                $l_i = j$

    comment: update centroids
    for $j \leftarrow 1$ to $k$
      do
        comment: $j^{th}$ cluster mean value
        $s \leftarrow 0$
        $n \leftarrow 0$
        for $i \leftarrow 1$ to $N$
          do
            if $l_i = j$
              then
                $s = s + \mathbf{x}_i$
                $n = n + 1$
        $\mathbf{m}_j^{(1)} = s/n$

return $(l_1, l_2, \cdots l_N)$
```

It is worth mentioning that the K-Means converges incredibly fast. This is one of the reasons why it is so popular, aside its simplicity. The algorithm can be cast as a *Newton-Raphson*-like algorithm and a calculation of its hessian shows that its convergence is practically superlinear - what forbids it to be so are the discontinuities of the energy function E. An intuitive argu-

ment observed superlinear convergence can be seen at [10], some solid results about the worst-case running time are demonstrated on [4], [15].
We remind that once the data set is assumed to have n points to be grouped into k clusters, there would have k^n possibilities for cluster configurations (including empty clusters), although it is a finite number, it gets huge already for small k and n, hence it is unpractical to search for every clustering possibility to find the *global minimum* of E is reached. As a matter of fact, this search problem is *NP-hard* [23], so we do not expect to solve it in polinomial time. One could design a stochastic algorithm that makes this search randomly and selects the best result. Even this strategy does not sound sensible, and the reason lies in the loss of memory between two iterations - the K-Means delivers good results because it minimizes the cost function gradually, so that even if the inital centroids are randomly selected, during the course of the algorithm the updates for the centroids capture the natural structure associated to the data set. As a drawback, the K-Means does not guarantee to find the *global minimum*, but we can get around it partially by running the algorithm many times and selecting the best clustering (*i.e.*, that with the smaller energy E) and also selecting carefully the initial centroids [5]

2.2.1 Behind the algorithm convergence

We can rewrite Equation 2.4 for the centroids at the $(n + 1)^{th}$ iteration step in a way that makes explicit their dependence on the centroid at the previous iteration step. For this sake we define the function $\varphi = (\varphi_1, \varphi_2, \cdots \varphi_k)$ by the relation

$$\mathbf{m}_i^{(n+1)} = \frac{\sum\limits_{\mathbf{x}\in D} r_{\mathbf{x}}^{(n)}(i)\ \mathbf{x}}{\sum\limits_{\mathbf{x}\in D} r_{\mathbf{x}}^{(n)}(i)} =: \varphi_i(\mathbf{m}_1^{(n)}, \cdots, \mathbf{m}_k^{(n)})$$

$$\text{for} \quad i = 1, 2, \cdots k$$

In this notation the iterative procedure is actually the calculation of a sequence starting at $\mathbf{M}^{(0)}$ and recursively given by the relation

$$\begin{aligned} \mathbf{M}^{(n+1)} &= \varphi\left(\mathbf{M}^{(n)}\right) \\ \varphi(\mathbf{M}^{(n)}) &= \left(\varphi_1\left(\mathbf{m}_1^{(n)}\ ,\ \cdots, \mathbf{m}_k^{(n)}\right), \cdots, \varphi_k\left(\mathbf{m}_1^{(n)}, \cdots, \mathbf{m}_k^{(n)}\right)\ \right) \end{aligned} \tag{2.6}$$

We aim to show that associated to the Sequence 2.6 there is a function $E = E(\mathbf{M})$ such that $E(\varphi(\mathbf{M}^{(n+1)})) \leq E(\mathbf{M}^{(n)})$ for all n. Such a function is said to be a *descent* of the sequence $\{\mathbf{M}^{(n)}\}$ and will be taken as the objective function that the K-Means seeks to minimize. Consider a k-tuple of centroids $\mathbf{M} = (\mathbf{m}_1, \mathbf{m}_2, \cdots \mathbf{m}_k)$. We define the classification function for this configuration of centroids as

$$Y(\mathbf{x}) = \underset{j\in\{1,2,\cdots k\}}{\text{argmin}} \quad \|\mathbf{x} - \mathbf{m}_j\|_{L^2}$$

Then, we define the descent function, also called the “energy” or the “cost function”, as a function $E\colon \mathbf{M} \mapsto \mathbb{R}$ given by

$$E = \sum_{\mathbf{x} \in D} \|\mathbf{x} - \mathbf{m}_{Y(\mathbf{x})}\|_{L^2}^2 \tag{2.7}$$

(where we have omitted the argument “$\mathbf{M}$” in $E(\mathbf{M})$).
The clusters for this specific k-tuple of centroids are

$$C_i = \{\mathbf{x} \in D \mid Y(\mathbf{x}) = i\}$$

in a way that

$$Y(\mathbf{x}) = i \iff \mathbf{x} \in C_i$$

therefore, the objective function can be rewritten as

$$E = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{m}_i\|_{L^2}^2 \tag{2.8}$$

The Expression 2.8 suggests us to define the function $E_{C_i}(\mathbf{z}) = \sum\limits_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{z}\|_{L^2}^2$ that we minimize by equating its partial derivatives to zero:

$$0 = \frac{\partial E_{C_i}(\mathbf{z})}{\partial z_i} = \sum_{\mathbf{x} \in C_i} \frac{\partial}{\partial z_i} \sum_{j=1}^{d} (x_j - z_j)^2 = \sum_{\mathbf{x} \in C_i} -2(x_i - z_i)$$

$$\therefore \quad \sum_{\mathbf{x} \in C_i} x_i = \sum_{\mathbf{x} \in C_i} z_i = |C_i| z_i$$

This proves the following lemma

Lemma 2.2.1. *Given a cluster C (or any other set of points), the minimum of the function $E_C(\mathbf{z}) = \sum\limits_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{z}\|_{L^2}^2$ is reached at*

$$\mathbf{z} = \frac{1}{|C|} \sum_{\mathbf{x} \in C} \mathbf{x}$$

That is the reason why the centroid in the second step of each iteration of the original K-Means algorithm is the cluster mean vector.
Let us prove the following

Lemma 2.2.2. *During the course of the K-Means algorithm, the energy function monotonically decreases.*

Proof. Let $\mathbf{m}_1^{(n)}, \mathbf{m}_2^{(n)}, \cdots \mathbf{m}_k^{(n)}, C_1^{(n)}, C_2^{(n)}, \cdots C_k^{(n)}$ denote the centroids and the clusters at the start of the n^{th} iteration.
The first step of the n^{th} iteration assigns each data point to its closest centroid through the classification function $Y(\mathbf{x}) = \underset{j\in\{1,2,\cdots k\}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{m}_j^{(n)}\|_{L^2}$ and we update the clusters

$$C_i^{(n+1)} = \{\mathbf{x} \in D \mid Y(\mathbf{x}) = i\}$$

A point $\mathbf{x}$ belongs to $C_i^{(n)}$ if only if in the previous iteration $\|\mathbf{x} - \mathbf{m}_i^{(n-1)}\|_{L^2} \leq \|\mathbf{x} - \mathbf{m}_j^{(n-1)}\|_{L^2}$ for every $j = 1, 2, \cdots k$. Because Y seeks to minimize the L^2 distances, we have

$$\|\mathbf{x} - \mathbf{m}_{Y(\mathbf{x})}^{(n)}\|_{L^2}^2 \leq \|\mathbf{x} - \mathbf{m}_i^{(n)}\|_{L^2}^2$$

Every $\mathbf{x} \in D$ belongs to only one cluster $C_i^{(n)}$ at each iteration step, therefore summing the above relation over all data points in D gives

$$\sum_{\mathbf{x}\in D} \|\mathbf{x} - \mathbf{m}_{Y(\mathbf{x}))}^{(n)}\|_{L^2}^2 \leq \sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j^{(n)}} \|\mathbf{x} - \mathbf{m}_j^{(n)}\|_{L^2}^2$$

For the above calculation, we partitioned the summation on the right as $D = \cup_j \, C_j^{(n)}$. Now we partition the left summation as $D = \cup_j \, C_j^{(n+1)}$, implying

$$\sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j^{(n+1)}} \|\mathbf{x} - \mathbf{m}_j^{(n)}\|_{L^2}^2 \leq \sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j^{(n)}} \|\mathbf{x} - \mathbf{m}_j^{(n)}\|_{L^2}^2 \tag{2.9}$$

Finally, using Lemma 2.2.1, $\mathbf{m}_i^{(n+1)}$ is by definition the minimum of the function

$$E_{C_j^{(n+1)}}(\mathbf{z}) = \sum_{\mathbf{x}\in C_j^{(n+1)}} \|\mathbf{x} - \mathbf{z}\|_{L^2}^2$$

hence for each j, $E_{C_j^{(n+1)}}(\mathbf{m}^{(n+1)}) \leq E_{C_j^{(n+1)}}(\mathbf{m}^{(n)})$ which in combination with Equation 2.9 gives

$$\sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j^{(n+1)}} \|\mathbf{x} - \mathbf{m}_j^{(n+1)}\|_{L^2}^2 \leq \sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j^{(n)}} \|\mathbf{x} - \mathbf{m}_j^{(n)}\|_{L^2}^2$$

which we can restate concisely as

$$E(\mathbf{M}^{(n+1)}) \leq E(\mathbf{M}^{(n)})$$

□

We say the sequence $(\mathbf{M}^{(n)})_{n=0,1,2,\cdots}$ defined by 2.6 enters an r-cycle starting at n if $\mathbf{M}^{(n+r)} = \mathbf{M}^{(n)}$ and $\mathbf{M}^{(i)} \neq \mathbf{M}^{(j)} \ \ \forall \ \ n \leq i < j < n+r$.
As a consequence of the last lemma we have the

Corollary 2.2.2.1. *If the sequence $(\mathbf{M}^{(n)})_{n=0,1,2,\cdots}$ enters an r-cycle at n, then $E(\mathbf{M}^{(i)}) = E(\mathbf{M}^{(n)}) \ \ \forall \ \ i \geq n$.*

Proof. If this is not the case, then, there is a $i > n$ such that $E(\mathbf{M}^{(i)}) \neq E(\mathbf{M}^{(n)})$. As i belongs to a cycle which is equal to the first one, we can consider that $n < i < n+r$.
And using the lemma above we have

$$E(\mathbf{M}^{(n)}) < E(\mathbf{M}^{(i)}) \leq E(\mathbf{M}^{(n+r)}) = E(\mathbf{M}^{(n)})$$

which gives in turn $E(\mathbf{M}^{(n)}) < E(\mathbf{M}^{(n)})$, namely a contradiction.

□

We now state a first, simpler convergence theorem for the Sequence 2.6.

Theorem 2.2.3. *Given a sequence $(\mathbf{M}^{(n)})_{0,1,2,\ldots}$, if $E(\mathbf{M}) \neq E(\mathbf{N}) \ \ \forall \ \ \mathbf{M}, \mathbf{N} \in \{\mathbf{M}^{(n)} \ , \ \ n = 0, 1, \cdots\}$, then, the sequence defined by 2.6 will reach a fixed point in finite time.*

Remark. *Clearly if a fixed point is reached at the n^{th} iterarion,*

$$\mathbf{M}^{(n+1)} = \varphi(\mathbf{M}^{(n)}) = \mathbf{M}^{(n)} \qquad \therefore \qquad \mathbf{M}^{(n+2)} = \varphi(\mathbf{M}^{(n+1)}) = \varphi(\mathbf{M}^{(n)}) = \mathbf{M}^{(n)}$$

And by the induction principle,

$$\mathbf{M}^{(n+p)} = \mathbf{M}^{(n)} \ \ \forall \ \ p \geq 0$$

Proof. The data set is $D = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_N\}$ and at each step k clusters are formed, so that there are k^N possible clustering configurations, meaning that there are only k^N different values for the $\mathbf{M}^{(n)}$'s. Though that usually gives a huge number, it is finite.
By hypothesis, and using Lemma 2.2.2, the objective function is strictly decreasing, thus there are no r-cycles with $r > 1$, because if this was the case we would have the contradiction

$$E(\mathbf{M}^{(n)}) = E(\mathbf{M}^{(n+r)}) < E(\mathbf{M}^{(n)})$$

As the value for $\mathbf{M}^{(n+1)}$ depends only on the $\mathbf{M}^{(n)}$ (through the function φ), if at some point the iteration gives $\mathbf{M}^{(n)} = \mathbf{M}^{(n+r)}$, this would necessarily incur in a r-cycle, and by the previous observation, r should be necessarily 1. Thus there are no repetitions for two non-consecutive values for $\mathbf{M}^{(\cdot)}$.
If each new value for $\mathbf{M}^{(n)}$ makes the objective function decrease strictly and there cannot be any non-consecutive repetitions, then, at some n equal at most to k^N there should be a consecutive repetition (a 1-cycle), *i.e.*,

$$\mathbf{M}^{(n+1)} = \varphi(\mathbf{M}^{(n)}) = \mathbf{M}^{(n)}$$

□

This result only accounts for when the centroids generated by each iteration follow the strict inequality for the energy function. We could guess if there are situations where two different clusterings have the same energy. The geometric intuition says that this would be an odd event, so we conjecture that the set of configurations $D = (\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_N)$ which verifies this odd situation has null measure.

Conjecture 2.2.1. *Given N and k, consider the set $\mathbb{R}^{dN}$ of all possible configurations for data points $D = (\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_N)$, and for each such configuration, consider the set of all its clusterings, represented by their centroids. We conjecture that the set of configurations $D \in \mathbb{R}^{dN}$ such that there are at least two clusterings $\mathbf{M}$ and $\mathbf{M}'$ with $E(\mathbf{M}) = E(\mathbf{M}')$ has zero Lebesgue measure. If we define this set as*

$$A = \{D = (\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_N) \in \mathbb{R}^{dN} \quad | \quad E(\mathbf{M}) = E(\mathbf{M}') \quad \textit{for some two clusterings of } D\}$$

then, $m(A) = 0$, where m is the standard Lebesgue measure defined for $\mathbb{R}^{dN}$.

The proposed conjecture was stated for the sake of completeness of this chapter. In fact there is no need to try proving it because a complete proof of convergence is given in chapter 4, in an optimization framework.

2.3 Soft K-Means

The K-Means algorithm, despite its simplicity, has the drawback of not being able to detect shape peculiarities of clusters, like cigar shapes, or huge radius differences. Of course, data sets possessing such special characteristics will make the algorithm perform badly, as is displayed in 2.1.

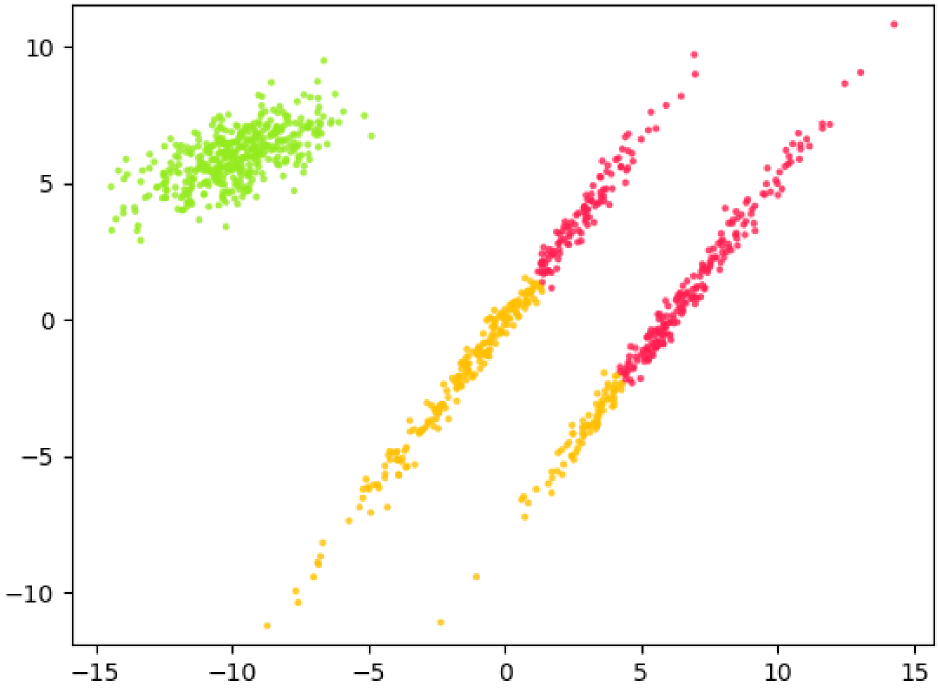


Figure 2.1: The two cigar-shaped clusters are so alongated that the K-means cannot locate properly the clusters' centers thus producing a bad clustering.

In order to deal with these kind of data, which are by no means rare, we need to enhance our algorithms, by introducing 'new degrees of freedom' on their shape detection capabilities. We will see further on chapter 3 section 3.7.1 that the K-Means is a limit of a mixture of gaussians all sharing the same radius,thus performing poorly when the actual data presents natural clusters of varying shapes and highly distinct sizes. The root of the presented K-Means drawback lies in its 'hardness', hence it is natural to modify the original algorithm by introducing probability concepts in the assignment of labels. In the following we discuss one enhancement for the K-Means, the *soft K-Means* which will introduce a new parameter and will allow to capture some subtle shape peculiarities (for a nice discussion see Mackay, [21]). Despite introducing another degree of flexibility for shape detection, the *soft K-Means* still will not be able to detect reliably extreme shapes like those in 2.1. However will motivate a rich framework to work on, which is developed in chapter 3.
Let us recall the definition 2.4 for the responsibility function, omitting for simplicity the superscript (n) (r should be recalculated at every iteration):

$$r_{\mathbf{x}}(i) = \delta(i, Y(\mathbf{x})) = \begin{cases} 1, & \text{if} \quad \underset{j \in \{1,2,\cdots k\}}{\text{argmin}} \quad d(\mathbf{x}, \mathbf{m}_j) = i \\ 0, & \text{if} \quad \underset{j \in \{1,2,\cdots k\}}{\text{argmin}} \quad d(\mathbf{x}, \mathbf{m}_j) \neq i \end{cases}$$

Where Y is the classification function $Y(x) = \underset{j \in \{1,2,\cdots k\}}{\text{argmin}} \quad d(\mathbf{x}, \mathbf{m}_j)$.
So, given a point $\mathbf{x}$, there is one and only one possible label for it, among $\{1, 2, \cdots k\}$. Also, given a set of centroids $(\mathbf{m}_1, \mathbf{m}_2, \cdots \mathbf{m}_k)$, the state space can be divided into regions

$$S_i = \{\mathbf{x} \in \mathbb{R}^{\mathrm{d}} \quad | \quad Y(\mathbf{x}) = i\} \quad \text{for} \quad i = 1, 2, \cdots k$$

The S_i regions are known as Voronoi sets. As pictured in Figure 2.2, the K-Means makes a geometric assignment for each data point, dividing the plane into clear-cut non-overlapping regions for each label. In this sense, K-Means is said to be a *hard clustering* algorithm, and this *hardness* is reflected in the straight line boundaries of each *Voronoi set.*

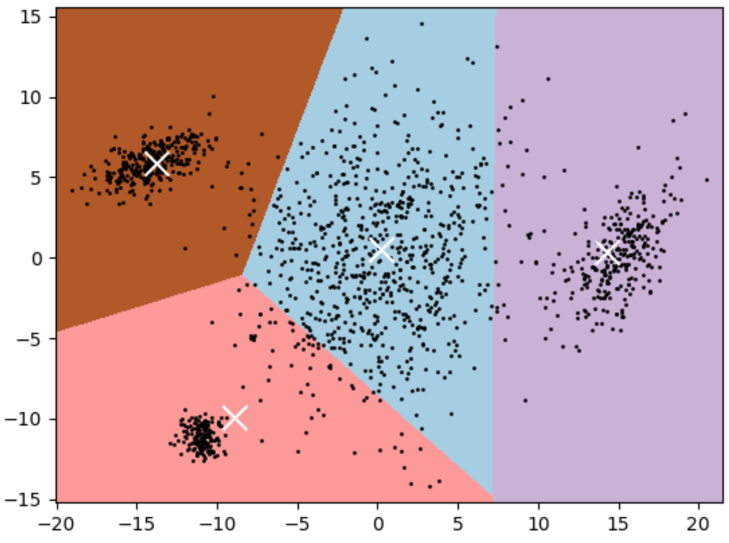


Figure 2.2: Geometric regions delimited by clusters boundaries, so-called *Voronoi sets.* The white crosses are the clusters centroids. Notice that the plane is divided into four more or less equal sized regions, but the central cluster is huge in comparison to the smaller just below, clearly this affected the result, because the smaller cluster centroid is shifted from its 'real' center.

The K-Means *hardness* entails a certain inability to detect properly cigar-shaped clusters, or uneven sized clusters. Also, K-Means tends to group the points in equally sized clusters (the reason for that will be understood at Chapter 3). Figures 2.3 and 2.4 give examples where the *hard* K-Means fails to properly classify the data points.

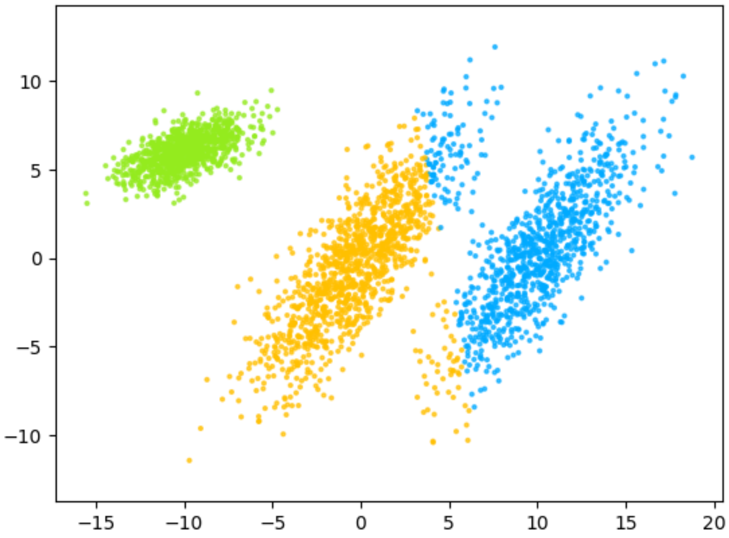


Figure 2.3: Two alongated clusters are displayed, their closeness makes it difficult for the k-means to adjust the Voronoi sets properly.

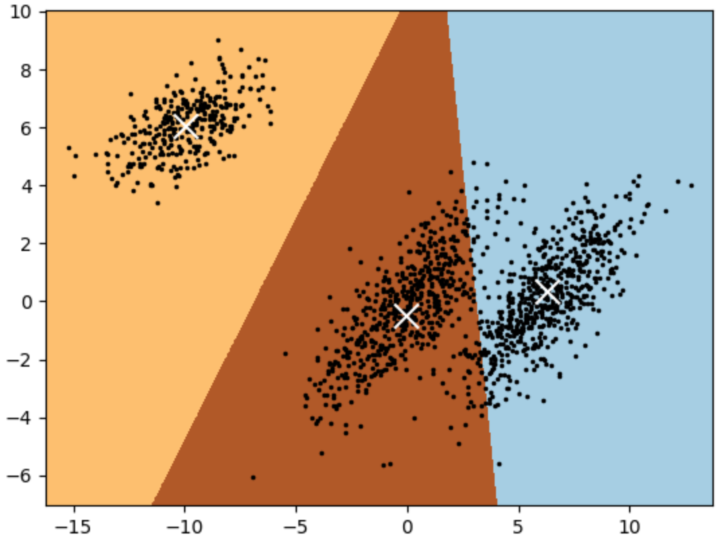


Figure 2.4: The Voronoi sets are displayed together with the clusters centers (whote crosses). The hardness of the assignments reflects in the polygonal geometry for the Voronoi sets, which makes difficult to fit too close cigar-shaped blobs.

A criticism to K-Means is that it does not leave space for a probabilistic analysis of the classification problem we are dealing with. A probabilistic analysis entails a richer framework for algorithm design. And most importantly, a probabilistic analysis better reflects the empirical nature of data: there is always some degree of imprecision related to the data classification into k different labels, often data points are difficult to distinguish, for example, when they lie next to the borders of the Voronoi sets. Consider the example discussed in Section 2.1 at the beginning of the chapter, where we had two distinct groups of clients, the creditors and the debtors. There could exist some situations where clients could easily move to the other cluster - as for example critical creditors who actually are quite close to a negative credit status. We might want to identify these clients with high cluster mobility in order to evaluate risk exposure. On the contrary K-Means would classify them in such a *hard*, strict, way that we would not realize they are in a risky situation, that is, they lie next to the border of a Voronoi set.
The K-means could be modified by specifying at every data point $\mathbf{x}$ its varying degrees of belonging to the each cluster, through a set of probabilities $p_1, p_2, \cdots p_k$ where p_i stands for the probability of $\mathbf{x}$ to fall in C_i. So instead of classifying each point with one label only - through a $\{0,1\}$ responsibility function - we could now give a probability distribution over all the labels, *softly* classifying the points. Thus, the responsibility function can be modified in a general way as

$$r_{\mathbf{x}}(i) = p_i(\mathbf{x}, \mathbf{M}, D)$$

Where $p_i(\mathbf{x}, \mathbf{M}, D)$ takes values in the interval $[0,1]$ and is seen as the probability of $\mathbf{x}$ being labelled by i given the centroids $\mathbf{M}$ and the others data points D. Of course, the generality of the the p_i's choice allows us to design better clustring methods regarding the clusters shape pecularities detection, once we can introduce new parameters to capture and fit such data local behaviours. But on the other side, introducing many new parameters could produce an *overfitting* of data, an issue we postpone for later in chapter 3. For now we propose - perhaps a bit out of the blue - the following choice for p_i:

$$r_{\mathbf{x}}(i) = p_i(\mathbf{x}, \mathbf{M}, D) = \frac{\mathrm{e}^{-\beta d(\mathbf{x}, \mathbf{m}_i)}}{\sum\limits_j \mathrm{e}^{-\beta d(\mathbf{x}, \mathbf{m}_j)}} \tag{2.10}$$

Readers familiar with statistical physics will recognize the Gibbs sample assignment of probabilities in 2.10. In this framework, β would be the inverse of the temperature - but what is 'temperature' in this system? Here we will not analyse this new algorithm from the analogy with statistical physics, such a development can be seen at [20]. We include a dedicated section for the soft K-Means motivating its definition and detail its convergence analysis in the next chapter.
We design a modified version of the *hard* K-Means by retaining the iterative procedures $i.$ and $ii.$ in 2.2, 2.5, respectively, but making use of this new responsibility function, thus the proposed algorithm will run the same as for the K-Means with the difference that the centroids are updated to a weighted sum over *all* data points in D. So we alternate at each iteration the two steps:

Assign: for each $\mathbf{x} \in D$, assign the new probabilities:

$$p_i^{(n+1)} = p_i(\mathbf{x}, \mathbf{M}^{(n)}, D) = \frac{\mathrm{e}^{-\beta d\left(\mathbf{x}, \mathbf{m}_i^{(n)}\right)}}{\sum\limits_j \mathrm{e}^{-\beta d\left(\mathbf{x}, \mathbf{m}_j^{(n)}\right)}}$$

Update: Update centroids $\mathbf{m}_i^{(n)}$ to:

$$\mathbf{m}_i^{(n+1)} = \frac{\sum\limits_{\mathbf{x} \in D} r_{\mathbf{x}}^{(n)}(i)\ \mathbf{x}}{\sum\limits_{\mathbf{x} \in D} r_{\mathbf{x}}^{(n)}(i)} = \frac{\sum\limits_{\mathbf{x} \in D} p_i^{(n+1)}\ \mathbf{x}}{\sum\limits_{\mathbf{x} \in D} p_i^{(n+1)}} \quad \text{for} \quad i = 1, 2, \cdots$$

A pseudocode of the soft K-Means is presented below. And a Python implementation is given in the appendix A.2.

Algorithm 2.3.1: K-MEANS(k, D, ϵ)

```
for $i \leftarrow 1$ to $k$
  do $\mathbf{m}_i^{(0)} \leftarrow$ *random value*

$\mathbf{M}^{(0)} \leftarrow (\mathbf{m}_1^{(0)}, \cdots \mathbf{m}_k^{(0)})$

$\mathbf{M}^{(1)} \leftarrow (\mathbf{0}, \cdots \mathbf{0})$

$(l_1, l_2, \cdots l_N) \leftarrow (0, 0, \cdots 0)$

while $\|\mathbf{M}^{(1)} - \mathbf{M}^{(0)}\| > \epsilon$
  do
    comment: calculate each $Y(\mathbf{x}_i)$
    for each $\mathbf{x}_i \in D$
      do
        $(p_{i1}, \cdots, p_{ik}) \leftarrow \left(\mathrm{e}^{-\beta d\left(\mathbf{x}_i, \mathbf{m}_1^{(0)}\right)}, \cdots, \mathrm{e}^{-\beta d\left(\mathbf{x}_i, \mathbf{m}_k^{(0)}\right)}\right)$
        $s \leftarrow 0$
        for $j \leftarrow 1$ to $k$
          do $s \leftarrow s + p_{ij}$

        $(p_{i1}, \cdots, p_{ik}) \leftarrow (p_{i1}/s, \cdots, p_{ik}/s)$

    comment: update centroids
    for $j \leftarrow 1$ to $k$
      do
        comment: $j^{th}$ cluster mean value
        $s \leftarrow 0$
        $\mathbf{m}_j^{(1)} \leftarrow \mathbf{0}$

        for $i \leftarrow 1$ to $N$
          do
            $s \leftarrow s + p_{ij}$
            $\mathbf{m}_j^{(1)} \leftarrow \mathbf{m}_j^{(1)} + p_{ij}\mathbf{x}_i$

        $\mathbf{m}_j^{(1)} \leftarrow \mathbf{m}_j^{(1)}/s$

return $(\mathbf{p}_1, \mathbf{p}_2 \cdots \mathbf{p}_N)$
```

(In the returned output above, $\mathbf{p}_i = (p_{i1}, p_{i2}, \cdots p_{ik})$ for each $i = 1, 2, \cdots N$. In the input, ϵ is a tolerance value for the stop criteria: the code stops when the difference (in a suitable matrix norm, for example, the *Frobenius norm*) between the matrices $\mathbf{M}^{(1)}$ and $\mathbf{M}^{(0)}$ becomes negligible, less than ϵ. Also, it could be necessary to begin with the first loop outside the *WHILE* statement, as happens in the Python implementation proposed in the appendix A).

Figure 2.5 ilustrates a soft k-Means Clustering accompanied to its Voronoi-like regions, whose borders are blurred (or, *fuzzy*) regions coloured in a varying mixture of each cluster's representative colour. In fact the Voronoi-like regions are not really much different than that of hard

K-Means, but the probabilistic framework gives us much more information about the data points than the purely geometric framework, specially in the blurred borders, also, this framework intuitively (as seen in the figure below) seems more natural to fit the data.

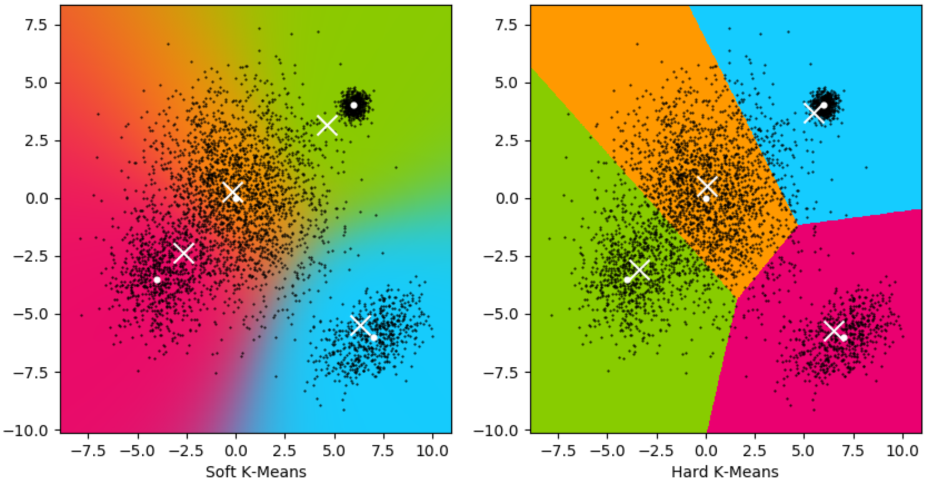


Figure 2.5: There are above four uneven sized clusters whose original centers are marked as round white circles, while the estimated centroids are marked in white 'X'.

Figures 2.6 and 2.7 give evidence that soft K-Means work better than the hard K-means. It is presented a set of points with alongated clusters some of which lying very close to each other. On the second set of points (with four natural clusters) the hard K-Means becomes unstable and group two natural cluster into one sole cluster, while divides another in two clusters. On the second set of images, we see that even the configuration where the K-Means worked well before, now it got stuck a bad clustering configuration, and in the set with 4 natural clusters the algorithm performed poorly. The soft K-Means instead appears to be unaffected, thus exhibiting greater stability than the hard K-Means.

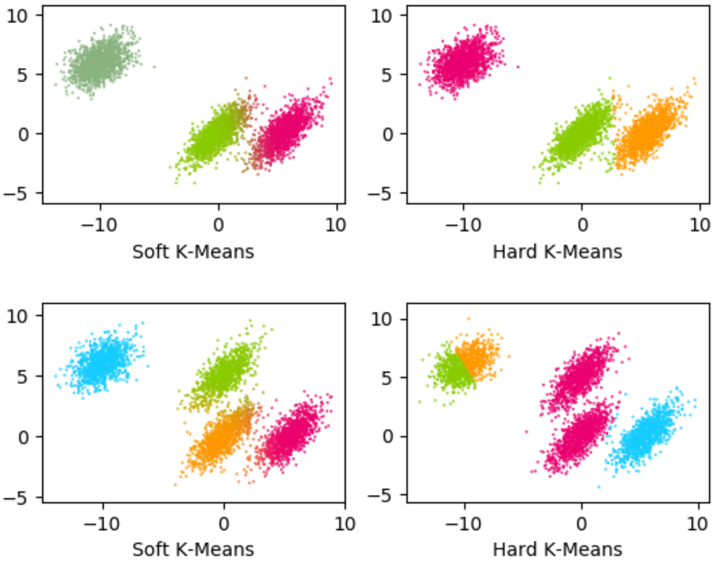


Figure 2.6: On the top a $k = 3$ clustering, below, $k = 4$, comparing as indicated, the *hard* versus *soft* K-Means. Evidence shows that the soft version of the algorithm is more robust against exotic configuratios like that verified in the bottom right, by running it many times.

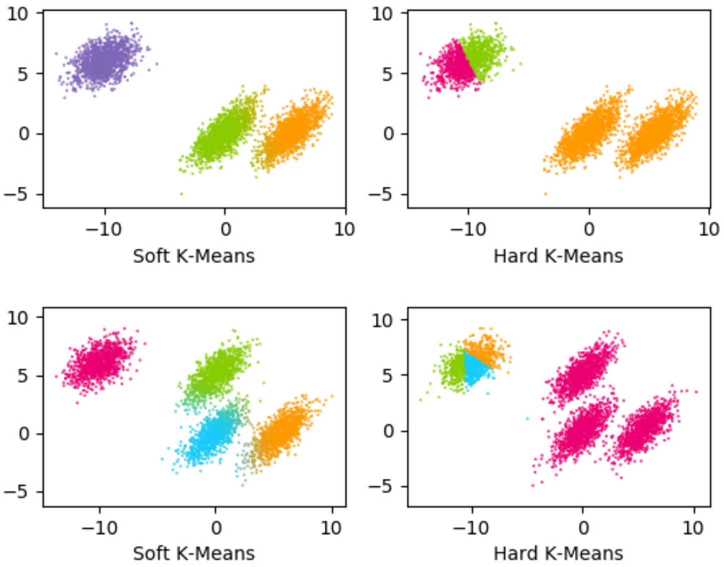


Figure 2.7: Above we see a very dramatic event on the top, two clusters are merged, and another on the bottom, three of them are assembled together, while the remaining one is divided by three. It is verified that soft K-Means is quite robust against this pathology, exactly because it accounts the varying degrees of belonging to each cluster.

The parameter β needs to be tuned. Different values for β are tested for the following data set in Figure 2.8, which also compares the results to a hard K-Means clustering. We notice that when β grows, the Voronoi-like sets become less blurred in its 'borders', and become very similar to the hard K-Means true Voronoi sets.

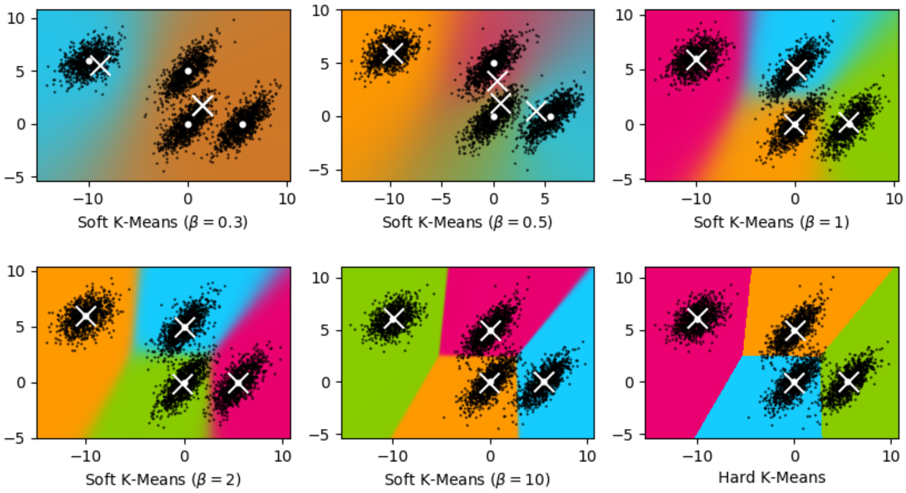


Figure 2.8: Different values for β are tested. Later we will interpret it as the inverse of the cluster radius, therefore, when it is small, like when $\beta = 0.3$, the three natural clusters coalesce (under the soft K-Means assignments) into three big clusters centered almost at the same location (so the mix of colors assume an uniform aspect). Notice as well that we recover the aspect of a Voronoi set when β gets big, here, above ~ 10.

Although the insertion of a new parameter β permits to identify more complex cluster shapes, or at least make the algorithm more robust to 'exotic' patterns, as shown in Figures 2.6 and 2.7, it is still not sufficient to account for the inner structure of each cluster separately. This issue is adressed in the next chapter where discuss another popular Clustering method, the *Gaussian Mixture Model*, which models a cluster as a Gaussian distribution around its centroid whose covariance specifies its shape, thus bringing more precision in the identification of clusters shapes. In the following figure we apply a *Gaussian Mixture Model* and compare its Voronoi-like region to both the hard and soft K-Means.

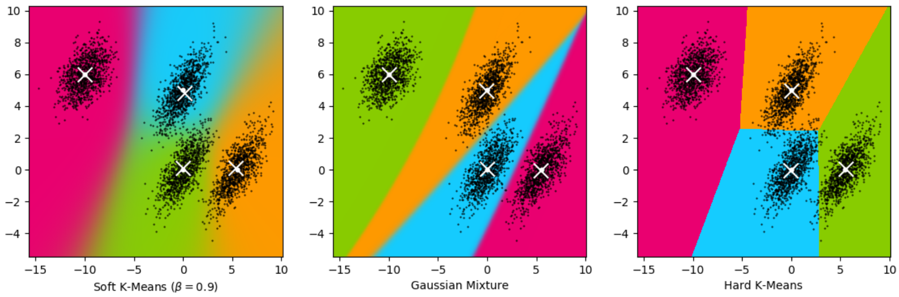


Figure 2.9: Comparison between three different methods - Soft K-Means, Gaussian Mixture, Hard K-Means. Notice that the Voronoi-like regions of Gaussian Mixture appear more suitable to the nature of clusters, notice also that the borders are blurred, though with good contrast.

Chapter 3

A probabilistic framework for clustering

3.1 General Mixture Models and Cluster Analysis background

A *Clustering* of a finite set $D = \{x_1, x_2, \cdots x_N\}$ is generally defined as a partition of the data set into k disjoint subsets $\{C_1, C_2, \cdots C_k\}$ (then $\dot{\bigcup}_{i=1}^{k} C_i = D$). A *Clustering procedure* is any process which returns as output one particular Clustering configuration. We interpret the elements belonging to a cluster C_i as 'similar' (where similarity can be represented by a distance metric $d(\cdot,\cdot)$), and elements belonging to different clusters as 'dissimilar'.

Notice that this is a possible - not the only - definition for clustering, and we can use many clustering procedures for studying the same data set. Besides, there exists a confusion between *clustering* and *clustering procedure.* The clustering algorithm generates as output a partition of the data, this is what generally people define as clustering. However, the partition in itself does not tell much about the nature of the data. Which clustering procedure works best on a specific data set depends on our prior information and assumptions about its intrinsic nature. As a consequence, there is no single universal tool for unsupervised learning, and we consider definition for clustering that depend on the nature of the data.

In this thesis we are going to define clustering as a component of a mixture model. We make the assumption that our data points are generated by random processes, in such a way we are dealing with problems that are probabilistic in nature. Clearly, this is one standpoint and it is not universal. However, it has the benefit of making sense of many algorithms, for example, we anticipate that we understand K-Means as a "limit" of the, so called, Gaussian Mixture Model. In the next subsection we analyze a general probabilistic Clustering procedure, the *Mixture Model*, with special attention to the particular case of *Gaussian Mixture Model*, simply called 'GMM'. Later on, we analyze how K-Means can be viewed as a limit case of a *GMM.* Finally, we analyse Clustering procedures as approximations to Mixture Models.

3.1.1 A trivial example

A mixture model is a weighted superposition of simpler probability distributions, representing our models. A random variable X can be generated by many possible models, each one having its own probability of generating the values of X.

For example, suppose a scientist wants to measure the weight of the participants of an experiment. Say that among them 65% are from São Paulo (SP) and the remaining 35% come from Rio de Janeiro (RJ). Now, we expect some differences over the distribution of weights, which we suppose is well approximated to a Gaussian, among citizens of these two Brazilian states. It might be that people in Rio are fatter than in São Paulo, and that the Gaussian for Rio population has a higher variance σ and higher mean μ.

As the scientist samples the participants, there will be at a time 35% of chance of selecting a 'carioca' (from Rio), in which case, he will verify a 'fatter' and broader gaussian in comparison to São Paulo, and 65% of observing a 'paulista' (from São Paulo), so observing the conditional probabilities

$$\begin{cases} p(weight = x|\text{``}carioca\text{''}) & = \frac{1}{\sqrt{2\pi\sigma_{RJ}^2}}\exp(-(x-\mu_{RJ})^2/2\sigma_{RJ}^2) \\ p(weight = x|\text{``}paulista\text{''}) & = \frac{1}{\sqrt{2\pi\sigma_{SP}^2}}\exp(-(x-\mu_{SP})^2/2\sigma_{SP}^2) \end{cases}$$

With the priors given by

$$\begin{cases} P(\text{``}carioca\text{''}) & = 35\% \\ P(\text{``}paulista\text{''}) & = 65\% \end{cases}$$

The probability (density) of finding a value $x \in \mathbb{R}$ for the weight, *i.e.*, the probability that a randomly chosen participant of the experiment weights x (kg), is given by

$$p(weight = x) \equiv p(x) = p(x|\text{``}carioca\text{''})P(\text{``}carioca\text{''}) + p(x|\text{``}paulista\text{''})P(\text{``}paulista\text{''})$$

$$p(x) = 0.35.\frac{1}{\sqrt{2\pi\sigma_{RJ}^2}}\exp(-(x-\mu_{RJ})^2/2\sigma_{RJ}^2) + 0.65.\frac{1}{\sqrt{2\pi\sigma_{SP}^2}}\exp(-(x-\mu_{SP})^2/2\sigma_{SP}^2)$$

where for brevity we omitted the '*weight* ='.

3.1.2 The full Mixture Model

Starting from this trivial example let us construct a more general mixture model. In the example, the random variable could have been generated by two different mechanisms or models - like a gaussian characterizing weights of Rio inhabitants and a different function for weight in São Paulo - each with their specific probabilities.

Let us consider a random variable X generated by k different mechanisms, each one modelled by a probability density $\rho(x|\theta_i)$, where θ_i , $i = 1, 2, \cdots k$ are the parameters of the chosen models. At a time, the random choice of the generation mechanism is described by an auxiliary random variable denoted M (standing for 'mechanism') taking values in $\{1, 2, \cdots k\}$

and distributed according to *a priori* given probabilities $\{\pi_1, \pi_2, \cdots \pi_k\}$, thus, every time X is sorted, the probality of it being produced by i^{th} mechanism is given by $P(M = i) = \pi_i$.
A mixture model describes the random variable X that produces the observed outcomes $x_1, x_2, \cdots x_n$ making use of the unobserved auxiliary variable M that indicates the mechanism by which X is generated at a time. In general, the probability that an outcome of X belongs to a set $A \subset \mathbb{R}^{\mathrm{d}}$ *and* was generated by mechanism i is computed as

$$P(\ X \in A,\ M = i\ ) = P(\ M = i\ )P(\ X \in A \mid M = i\ ) = P(\ M = i\ ) \int_A \rho(x|\theta_i)dx$$

Where $P(\ X \in A \mid M = i\ ) = \int_A \rho(x|\theta_i)dx$ is the probability that X values fall into A, once it is sorted by the i^{th} generation mechanism. We aim to calculate the probability distribution for X, so we compute

$$P(X \in A) = \sum_i^k P(\ X \in A,\ M = i\ ) = \sum_{i=1}^k P(\ M = i\ )P(X \in A \mid M = i) =$$

$$= \sum_{i=1}^k \pi_i \int_A \rho(x|\theta_i)dx = \int_A \sum_{i=1}^k \pi_i \rho(x|\theta_i) \equiv \int_A \rho(x)dx$$

The above equation tells us that X is distributed according to the probability distribution function, *pdf*, $\rho(x) = \sum_{i=1}^k \pi_i \rho(x|\theta_i)$.
Because the mixture of models is defined through the parameters θ_i and π_i, we change slightly the notation to emphasize this fact and write $\rho(x) = \rho(x|\theta, \pi)$ (the number of components of this model is itself a parameter, but is represented in this notation by the sizes of θ and π).
Often, the models and the number of components are inferred *a priori*, however the weights of each model, the values of π_i, are generally not known and have to be inferred. We will discuss a procedure for inference of the weights called the *Expectation Maximization* algorithm.
A special case of a mixture of models is when each component $\rho(x|\theta_i)$ is a gaussian distribution. This model is called *Gaussian Mixture Model* (GMM) and is widely used as an approximation for the actual *pdf* of a given random variable X.

3.2 Mixture Model is a type of Soft Clustering

In this section we explore Mixture Models in depth. Suppose we are observing the outcomes of a random variable X, which compose our data $D = \{x_1, x_2, \cdots x_n\}$, where $x_i \in \mathbb{R}^d$, and let us consider the case where the generation of X is best explained by a mixture model of k components, and assume that both π and θ values were already inferred. Thus, X is modeled by the *pdf*

$$\rho(x) = \sum_{i=1}^k \pi_i \rho(x|\theta_i)$$

We denote $p(x|i) := \rho(x|\theta_i)$, $p(x) := \rho(x)$, $p(x,i) := P(X = x,\ M = i)$ (an abuse of notation, p is the compound probability associated to the pair (X, M), and satisfies

$$P(X \in A,\ M \in B) = \sum_{i \in B} \int_A p(x,i)dx$$

Similarly $p(i|x) \equiv P(M = i|X = x)$. We calculate

$$p(i|x) = \frac{p(x,i)}{p(x)} = \frac{p(x|i)p(i)}{p(x)} = \frac{p(x|i)p(i)}{\sum_{l=1}^{k} p(x|l)p(l)} = \frac{\pi_i \rho(x|\theta_i)}{\sum_{l=1}^{k} \pi_l \rho(x|\theta_l)} \tag{3.1}$$

$p(i|x)$ is the probability that the mechanism that generated a given x is the i^{th} model $\rho(\ \cdot\ |\theta_i)$. $\sum_{i=1}^{k} p(i|x) = 1$, thus we interpret the conditional probabilities $p(i|x)$ as soft assignments of labels for every x. Having this in mind, one can view a mixture model as defining a soft clustering for every data set $D = \{x_1, x_2, \cdots x_n\}$. Hence, if we estimate somehow the parameters that best fit the data D, we can calculate such soft assignment. The advantage of using Mixture Models to cluster data is that we introduce the parameters θ_i, $i = 1, 2, \cdots k$ that can be tuned to fit specifically every cluster special shape. But how do we estimate π and θ? One method is the *Expectation Maximization*, which is described in the next subsection.

3.3 Expectation Maximization

The algorithm for expectation maximization relies heavily on the Jensen inequality, for which the proof and definitions are left to the Appendix B:

Theorem 3.3.1. *Let f be a convex function, and let X be a random variable. Then:*

$$E[f(X)] \geq f(E[X]) \tag{3.2}$$

Moreover, if f is strictly convex, then $E[f(X)] = f(E[X])$ holds true if and only if $P(X = E[X]) = 1$ (i.e., $X = E[X]$ except for a set of null measure).

Our approach is very similar (but with different notations) to the excellent references [25]. The background is the following: we have two variables, X and M, and two sets of parameters, $\pi = (\pi_1, \pi_2, \cdots \pi_k)$ and $\theta = (\theta_1, \theta_2, \cdots \theta_k)$, which we denote collectively as $\phi = (\pi, \theta)$. Recall that our goal is to find the mixture model that best fits the data set $D = \{x_1, x_2, \cdots x_n\}$. We make use of the *MLE* estimators (already discussed in Chapter 1, Section 1.5) to find the best fit of ϕ, therefore we calculate

$$\phi_{MLE} = \underset{\phi}{\operatorname{argmax}}\ p(D|\phi) = \underset{\phi}{\operatorname{argmax}}\ l(\phi) \tag{3.3}$$

Where

$$p(D|\phi) = \rho(X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n|\phi) = \prod_{i=1}^{n} p(x_i|\phi)$$

$$l(\phi) = \ln p(D|\phi) = \ln \prod_{i=1}^{n} p(x_i|\phi) = \sum_{i=1}^{n} \ln p(x_i|\phi)$$

The maximization is over the observed values for the x's, while the values for M are unobserved and remain implicit in the definition for l - they are said to be *latent variables.* We make the unobserved latent variables explicit in the above formula:

$$l(\phi) = \sum_{i=1}^{n} \ln \sum_{z=1}^{k} p(x_i, z|\phi) \tag{3.4}$$

Where $p(x_i, z) = P(X = x, M = z)$. We also assumed $\mathbf{M}$ take values in $\{1, 2, \cdots k\}$ motivated by the K-Means application, but the following discussion need not to be restrict solely on finite discrete latent variables, thus we assume z could take values in a continuum as well, and for coherence we substitute the summation symbol $\sum_{z=1}^{k}$ by the more general $\sum_z$ meaning that we integrate through the (conditional) probability measure associated to $\mathbf{M}$ (in the case of a discrete variable it is simply the summation of probabilities, for continuous latent variables, $\sum_z = \int dp(z)$).

In general we do not expect to solve Equation 3.3 analytically, hence we would better find an approximate solution through algorithmic procedures. In a classic 1977 paper by Arthur Dempster, Nan laird and Donald Rubin [1], the referred authors proposed an iterative method called *Expectation-Maximization* (EM), for finding the maximimum likehood estimates of general statistical models possessing unobserved random variables. The EM method produces a sequence of estimates $\{\phi^{(t)}\}_{t=0,1,\cdots}$ that makes $\{\ l(\phi^{(t)})\ \}_{t=0,1,\cdots}$ to be a monotonic increasing sequence

$$l(\phi^{(0)}) \leq l(\phi^{(1)}) \leq \cdots \leq l(\phi^{(t)}) \leq l(\phi^{(t+1)}) \leq l(\phi_{MLE})$$

through iterations that alternate between performing an expectation (E) step and a maximization step (M). At the E step the expectation of the likelihood 3.4 is evaluated using the current estimate $\phi^{(t)}$ for the parameters, following the maximization (M) step, which updates $\phi^{(t)}$ to $\phi^{(t+1)}$ maximizing the expected likelihood found on the E step. Then $\phi^{(t+1)}$ is used to determine the distribution of the latent variables in the next E step. We study in detail these two steps further.

The EM is advantageous when knowledge of the latent variable facilitates the calculation of $l(\phi)$ in terms of the observed variables and vice-versa. The algorithm maximizes $l(\phi)$ by repeatedly constructing a lower bound for $max\{l(\phi)\} = l(\phi_{MLE})$ with the previous estimate for ϕ fixed (the E-step) and then optimizes that lower bound, assuming that the M (the latent variable) was observed. We proceed detailing each of the two steps.

3.3.1 The E step

At the beginning of every iteration, for each observation x_i, we will calculate a new special set of auxiliary distributions $Q_i\colon z \mapsto Q_i(z)$ over the range of M using the parameters from the previous step. This distributions will help us to bound below the maximum likelihood. In order to understand why we calculate this set of distributions $\{Q_i\}_{i=1,2,\cdots n}$ and what values should they assume we reason as follows.

Assume we are given such set of auxiliary distributions Q_i corresponding to each $x_i \in D$. By defnition of distribution we must have $\sum_z Q_i(z) = 1$. Then we construct for every i a new probability space associated to a random variable Y_i taking for each z the values

$$Y_i(z) = \frac{p(x_i, z|\phi)}{Q_i(z)}$$

and distributed according to $Q_i(z)$ (*i.e.*, $P_{Q_i}(Y_i = y) = Q_i(z)$ whenever $y = \frac{p(x_i,z|\phi)}{Q_i(z)}$). In this new space the expectation of Y_i is

$$E_{Q_i}[Y_i] = \sum_z Q_i(z)\frac{p(x_i, z|\phi)}{Q_i(z)}$$

We will make use of the Jensen Inequality 3.2 applied to the concave function $f(x) = \ln(x)$ (whose second derivative is $f''(x) = -1/x^2$), which gives

$$E_{Q_i}[\ \ln Y_i\ ] \leq \ln\left(E_{Q_i}[\ Y_i\ ]\right)$$

And the equality will hold if and only if

$$Y_i \equiv \textit{constant} \text{ (with probability 1)}$$

The random variable Y_i lives in this new probability space constructed over the given distribution Q_i - do not confuse with the probability space of (X, M) - and it was created only to allow us to use the Jensen inequality for $\{Q_i\}_i = 1, 2, \cdots n$. Now, once we fixed this set of distributions, we make the following calculations:

$$l(\phi) = \sum_{i=1}^{n} \ln p(x_i|\phi) = \sum_{i=1}^{n} \ln\left(\sum_z p(x_i, z|\phi)\right) =$$

$$= \sum_{i=1}^{n} \ln\left(\sum_z Q_i(z)\frac{p(x_i, z|\phi)}{Q_i(z)}\right) = \sum_{i=1}^{n} \ln\left(E_{Q_i}[Y_i]\right) \geq \tag{3.5}$$

$$\geq \sum_{i=1}^{n} E_{Q_i}[\ \ln Y_i\ ] = \sum_{i=1}^{n}\sum_z Q_i(z)\ln\left(\frac{p(x_i, z|\phi)}{Q_i(z)}\right)$$

The above calculations are valid for arbitrary distributions hence for each distribution Q_i over z values and for every ϕ, it is verified the inequality

$$l(\phi) \geq \sum_{i=1}^{n}\sum_z Q_i(z)\ln\left(\frac{p(x_i, z|\phi)}{Q_i(z)}\right) \tag{3.6}$$

which constitutes a lower bound for $l(\phi)$ given the set $\{Q_i\}_{i=0,1,\cdots}$ and the current guess for ϕ at the iteration considered. Still using the theorem of Jensen inequality, we could make the inequality hold with equality sign in 3.5 and 3.6 by plugging in a distribution such that Y_i becomes constant (with probability 1), *i.e.*, by requiring that

$$Y_i = c \quad \Leftrightarrow \quad \frac{p(x_i, z|\phi)}{Q_i(z)} = c\ , \forall \quad z$$

for some c not depending on z.
Therefore, among all the possible distributions Q_i we pick the only one satisfying the condition $Q_i(z) \propto p(x_i, z|\phi)$, and the constraint $\sum_z Q_i(z) = 1$ gives the proportionality factor as the inverse of $\sum_z p(x_i, z|\phi)$ and hence

$$Q_i(z) = \frac{p(x_i, z|\phi)}{\sum\limits_z p(x_i, z|\phi)} = \frac{p(x_i, z|\phi)}{p(x_i|\phi)} = p(z|x_i, \phi)$$

Q_i is the only distribution that makes the following condition true:

$$l(\phi) = \sum_{i=1}^{n} \sum_{z} Q_i(z) \ln \left(\frac{p(x_i, z|\phi)}{Q_i(z)} \right) \tag{3.7}$$

Notice that the values of $Q_i(z)$ depend on the current guess ϕ so suppose at the t^{th} iteration the guess for ϕ was calculated at the previous iteration to be $\phi^{(t-1)}$. We define the *E-step* as the assignment of the distributions Q_i's relying upon $\phi^{(t-1)}$ for every $i = 1, 2, \cdots n$ then we fix $Q_i(z)$ and determine the function $L(\phi) = \sum_{i=1}^{n} \sum_z Q_i(z) \ln \frac{p(x_i,z|\phi)}{Q_i(z)}$, that we will maximize at the M step. We summarize:

***E* step:** . *For each $i = 1, 2, \cdots n$, calculate and assign the values* $Q_i(z) = p(z|x_i, \phi^{(t-1)})$;
. *Determine the function* $L(\phi) = \sum\limits_{i=1}^{n} \sum\limits_z Q_i(z) \ln \frac{p(x_i,z|\phi)}{Q_i(z)}$

3.3.2 The M step

In the E step we calculated for every x_i the only distribution Q_i that makes 3.5 to hold the equality in condition 3.7, Q_i is exactly the conditional distribution $p(z|x_i, \phi^{(t-1)})$. Likewise the E step, at the *M-step* we maximize 3.7 and update the new guess for ϕ:

***M* step:** . *Update the parameters from $\phi^{(t-1)}$ to $\phi^{(t)}$ by maximizing $L(\phi)$:*
$\phi^{(t)} = \underset{\phi}{\operatorname{argmax}} \left\{ \sum_{i=1}^{n} \sum_z Q_i(z) \ln \frac{p(x_i,z|\phi)}{Q_i(z)} \right\}$

If l is bonded (at least locally), then by construction, $t - th$ iteration makes $l(\phi^{(t)})$ closer to $l(\phi_{MLE})$ as verified below using inequality 3.5:

$$l(\phi^{(t-1)}) = \sum_{i=1}^{n} \sum_{z} Q_i(z) \ln \frac{p(x_i, z|\phi^{(t-1)})}{Q_i(z)} \leq \sum_{i=1}^{n} \sum_{z} Q_i(z) \ln \frac{p(x_i, z|\phi^{(t)})}{Q_i(z)} \leq l(\phi^{(t)}) \leq l(\phi_{MLE})$$

$$\Rightarrow \quad l(\phi^{(t-1)}) \leq l(\phi^{(t)}) \leq l(\phi_{MLE})$$

Through these iterations, the successive values for $\phi^{(t)}$ produce the bounded monotone increasing sequence

$$l(\phi^{(0)}) \leq l(\phi^{(1)}) \leq \cdots \leq l(\phi^{(t)}) \leq l(\phi^{(t+1)}) \leq \cdots \leq l(\phi_{MLE})$$

So, convergence for some value $l^* = \lim_{t\to\infty} \phi^{(t)}$ is guaranteed, though not necessarily to global maximum, which is a drawback of this method, if we aim the exact MLE estimator. Although there are guarantees that $l(\phi^{(t)})$ converges, nothing is said about the parameters $\phi^{(t)}$ themselves, and even when they converge, we must know how fast they do, as the implementation should stop at finite time. We adress this discussion to the next section.
Finally we summarize the *EM* below and implement it next as a pseudo-code in 3.3.1 :

E step: assign $\quad Q_i(z) = p(z|x_i, \phi^{(t-1)})$;

M step: update ϕ from $\phi^{(t-1)}$ to $\phi^{(t)} = \underset{\phi}{\operatorname{argmax}} \left\{ \sum_{i=1}^{n} \sum_z Q_i(z) \ln \frac{p(x_i,z|\phi)}{Q_i(z)} \right\}$

A pseudocode for the EM algorithm is summarized next.

Algorithm 3.3.1: EXPECTATIONMAXIMIZATION(D, ϵ)

```
$\phi_0 \leftarrow$ random value
$l_0 \leftarrow \sum_i^n \ln p(x_i|\phi_0)$
$\Delta \leftarrow 2\epsilon$

while $\Delta > \epsilon$
do
    comment: E-step
    for each $x_i \in D$
      do $Q_i(z) \leftarrow p(z|x_i, \phi_0)$

    comment: M-step
    $\phi_1 \leftarrow \underset{\phi}{\operatorname{argmax}} \left\{ \sum_{i=1}^{n} \sum_z Q_i(z) \ln \frac{p(x_i,z|\phi)}{Q_i(z)} \right\}$

    $l_1 \leftarrow \sum_i^n \ln p(x_i|\phi_1)$

    $\Delta \leftarrow l_1 - l_0$
    $l_0 \leftarrow l_1$
    $\phi_0 \leftarrow \phi_1$

return $(\phi)$
```

3.4 A word on the EM convergence

We start noticing that the M step tries to maximize the expression

$$L(\phi) = \sum_{i=1}^{n} \sum_{z} Q_i(z) \ln \frac{p(x_i, z|\phi)}{Q_i(z)} \tag{3.8}$$

given the newly set parameters Q_i in the previous E step. Assume that ϕ lies in a set $\Omega \subset \mathbb{R}^m$. We maximize L over Ω, but there are in general many local maxima. Therefore, having set a given $\phi^{(t-1)} \in \Omega$ along with its L function (that depends on the newly calculated coefficients $Q_i(z)$) there can be many choices for $\phi^{(t)}$ in the M step. A good way to represent this set of choices is through a *point-to-set* map which associates to every $\phi \in \Omega$ (corresponding to $\phi^{(t-1)}$) the set of every possible updates $\phi' \in \Omega$ (corresponding to $\phi^{(t)}$) as defined below:

$$\begin{array}{cccc} \Phi\colon & \Omega & \to & \mathcal{P}(\Omega) \\ & \phi & \mapsto & \Phi(\phi) \end{array}$$

$$\Phi(\phi) = \{\phi' \in \Omega \mid L(\phi') \geq L(x),\ \forall\ x \in B(\phi', \epsilon)\}$$

Where L is calculated using $Q_i(z) = p(z|x_i, \phi)$ for $i = 1, 2, \cdots, n$ and $B(\phi', \epsilon)$ is a ball where ϕ' is a local maximum. With this definition, every sequence $\{\phi^{(0)}, \phi^{(1)}, \cdots, \phi^{(t)}, \phi^{(t+1)}, \cdots\}$ generated by the EM algorithm satisfies

$$\phi^{(t)} \in \Phi\left(\phi^{(t-1)}\right)$$

We state conditions to guarantee the convergence of $\{\phi^{(t)}\}$ towards local maxima in what follows. We rely on the following result by Zangwill [37], [3] which is proven in the appendix A.2.1. We will need the

Definition 3.4.1. *A point-to-set function Φ is closed at $x \in \Omega$ if whenever $x_n \to x$ and $y_n \to y$ with $y_n \in \Phi(x_n)$ implies $y \in \Phi(x)$. We say Φ is closed in the set $S \subset \Omega$ if it is closed at every $x \in S$.*

Theorem 3.4.1 (Zangwill Global Convergence Theorem)**.** *Let the sequence $\{x_n\}_{n\in\mathbb{N}^*}$ be generated by the point-to-set function $\Phi\colon \Omega \to \mathcal{P}(\Omega)$ through the rule $x_{n+1} \in \Phi(x_n)$ and an initial guess x_0. Let a solution set $\Gamma \subset \Omega$ be given and suppose that:*

i. All points x_n are contained in a compact set $S \subset \Omega$;

ii. Φ is closed for every $x \in \Omega/\Gamma$;

iii. There is a continuous function $E\colon \Omega \to \mathbb{R}$ such that

  (a) *If $x \notin \Gamma$, $E(y) > E(x)$ for all $y \in \Phi(x)$;*

  (b) *If $x \in \Gamma$, $E(y) \geq E(x)$ for all $y \in \Phi(x)$.*

Then all the limit points of $\{x_n\}_{n\in\mathbb{N}^*}$ are in the solution set Γ and $E(x_n)$ converges monotonically to $E(x)$ for some $x \in \Gamma$.

A function E satisfying the properties $iii.(a)$ and $iii.(b)$ is called a *descent function.* In general a so-called *descent method* implements an algorithm described by a point-to-set function that is specially designed to make a *descent function* E satisfy conditions $iii.(a)$ and $iii.(b)$. A classical example is the *gradient descent* [26]. K-Means itself is another example of a descent method.

Notice that the theorem does not garantee the existence of limit points to the sequence $\{x_n\}_{n\in\mathbb{N}^*}$, but in case the latter is bounded, there will be a subsequence converging to a solution $x \in \Gamma$. Back to the *Expectation Maximization* algorithm, the likelihood $l(\phi) = \ln\{p(D \mid \phi)\}$ is the *descent function*, and the *solution set* Γ is

$$\Gamma = \{\phi' \in \Omega \mid l(\phi') \geq l(x),\ \forall\ x \in B(\phi', \epsilon)\}$$

That is, Γ is the set of local maxima for $l = l(\phi)$ in Ω. By construction of the algorithm, $l(\phi^{(t)}) \leq l(\phi^{(t+1)})$ for all t - this guarantees satisfaction of condition $iii.$(b) and we assume the boundness, which can be verified through some condition like $l(\phi) \leq l(\phi_{MLE}),\ \forall\ \phi \in \Omega$ or the condition

$$\Omega_{\phi_0} = \{\phi \in \Omega \mid l(\phi) \geq l(\phi_0)\} \text{ is compact for any } \phi_0$$

The difficulty in the application of Zangwill's theorem is to show that the inequality $l(\phi^{(t)}) \leq l(\phi^{(t+1)})$ can be made strict when $\phi^{(t)}$ is not a local maximum - this would amount for satisfaction of condition $iii.(a)$ - and that the point-to-set function Φ is closed in Ω/Γ - condition $ii.$. Now, for each t, $\phi^{(t+1)}$ maximizes $L(\phi)$. If $\phi^{(t)}$ is not a maximum (*i.e.*, it does not belong to Γ), then if the condition $\phi^{(t+1)} > \phi^{(t)}$ is not satisfied, this would imply that the latter is in fact a local maximum (if not, there would be a direction in the manifold Ω where to maximize it strictly). The closedness property of Φ is normally associated to the continuity of $L(\phi)$, and depends on which case we are dealing.

This brief note is only to account that the sequence $\{\phi^{(t)}\}_{t\in\mathbb{N}^*}$ *EM* does indeed converge in most of the cases, but we will not focus in detailing this convergence, nor study its speed. For further details on such convergence properties, see [35]. We proceed next sections studying in detail the Soft K-Means and the Gaussian Mixture Model.

3.5 Soft K-Means explained

Consider a data point $x_0 \in D$ which is sorted from a mixture of distributions $\rho(\,\cdot\,|\theta_i)$ for $i = 1, 2, \cdots k$. Each distribution $\rho(\,\cdot\,|\theta_i)$ models a cluster. Recall expression 3.1 which gives the probability of such given x_0 having been generated by the i^{th} mechanism, or, in Cluster Analysis terms, its probability of belonging to the i^{th} cluster:

$$p(i|x_0, \phi) = \frac{\pi_i \rho(x_0|\theta_i)}{\sum_{l=1}^{k} \pi_l \rho(x_0|\theta_l)} \tag{3.9}$$

Where we made explicit the dependence of $p(i|x_0)$ on the parameters ϕ (encompassing θ's). Now compare the above expression to the *Soft K-Means* probability assignments as defined in 2.10

$$p_i(x_0, \mathbf{M}, D) = \frac{\mathrm{e}^{-\beta d(x_0, \mathbf{m}_i)}}{\sum\limits_j \mathrm{e}^{-\beta d(x_0, \mathbf{m}_j)}} \tag{3.10}$$

Where $\mathbf{M} = (\mathbf{m}_1, \mathbf{m}_2, \cdots \mathbf{m}_k)$ are the representative centroids of each cluster.
This comparison suggests that the k clusters are modeled each by a distribution with parameters β and $\mathbf{m}_i$ given by

$$\rho(x_0|\beta, \mathbf{m}_i) = \alpha\ \mathrm{e}^{-\beta d(x_0, \mathbf{m}_i)} \tag{3.11}$$

Where α is a normalization constant and each cluster should have weight $\pi_i = 1/k$ so that they sum up to 1 and are all equal. Indeed, suppose the data points are generated by the mixture with components like in 3.11 and equal weights

$$\rho(x|\beta, \mathbf{M}) = \sum_{j=1}^{n} \frac{1}{k} \alpha\ \mathrm{e}^{-\beta d(x_0, \mathbf{m}_i)}$$

Then, starting from 3.9 for this mixture we arrive at the *soft K-Means* probability assignment 3.10:

$$p(i \mid x_0,\ \beta,\ \mathbf{M}) = \frac{\frac{1}{k}\alpha\ \mathrm{e}^{-\beta d(x_0, \mathbf{m}_i)}}{\sum_{l=1}^{k} \frac{1}{k}\alpha\ \mathrm{e}^{-\beta d(x_0, \mathbf{m}_l)}} = \frac{\mathrm{e}^{-\beta d(x_0, \mathbf{m}_i)}}{\sum\limits_j \mathrm{e}^{-\beta d(x_0, \mathbf{m}_j)}} = p_i(x_0,\ \mathbf{M},\ D)$$

This shows that the soft K-Means *assignment* step is equivalent to the E step of the Expectation Maximization algorithm. We proceed showing that the *soft K-Means* update step is also equivalent to the M step for the special (and most important) case where $d(x, y) = \|x - y\|^2_{L^2}$.
Let us calculate precisely the E and M steps for this mixture at iteration t. For the E step, we should calculate the $Q_i(j)$'s for $j = 1, 2, \cdots k$, for each $i = 1, 2, \cdots n$ (i corresponding to the i^{th} data point), thus we calculate a matrix of coefficients $\left[\ q_{ij}\ \right] = \left[\ Q_i(j)\ \right]$ defined below:

$$q_{ij} = p\left(\ j \mid x_i,\ \beta,\ \mathbf{M}^{(t-1)}\right) = \frac{\mathrm{e}^{-\beta d\left(x_i,\ \mathbf{m}_i^{(t-1)}\right)}}{\sum\limits_j \mathrm{e}^{-\beta d\left(x_i,\ \mathbf{m}_j^{(t-1)}\right)}}$$

For the M step we calculate the following expression with $\rho(x|\beta, \mathbf{m}_i) = \alpha\ \mathrm{e}^{-\beta d(x_0, \mathbf{m}_i)}$ and $\pi_j =$

$1/k$:

$$\sum_{i=1}^{n}\sum_{j=1}^{k} Q_i(j)\ln\left\{\frac{p(x_i, j \mid \beta, \mathbf{M})}{Q_i(j)}\right\} = \sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij}\ln\left\{\frac{p(j)\rho(x_i|\beta, \mathbf{m}_j)}{q_{ij}}\right\} =$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij}\ln\left\{\frac{1/k\ \alpha}{q_{ij}}\ \mathrm{e}^{-\beta d(x_i, \mathbf{m}_j)}\right\} = \sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij}\left\{\ln\alpha - \ln(k\ q_{ij}) -\ \beta\ d(x_i, \mathbf{m}_j)\right\} =$$

$$= \left\{\sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij}\ln\alpha - \beta\ \sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij}\ d(x_i, \mathbf{m}_j)\right\} + constant \equiv$$

$$\equiv f(\beta, \mathbf{M}, (q_{ij})) + constant \tag{3.12}$$

We aim to maximize expression

$$\sum_{i=1}^{n}\sum_{j=1}^{k} Q_i(j)\ln\left\{\frac{p(x_i, j \mid \beta, \mathbf{M})}{Q_i(j)}\right\} = f(\beta, \mathbf{M}, (q_{ij})) + constant$$

which is done by equating to zero the derivatives in $\mathbf{m}_i$ of f:

$$0 = \nabla_{\mathbf{m}_l} f(\beta, \mathbf{M}^{(t)}, (q_{ij})) = -\beta\ \sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij}\ \nabla_{\mathbf{m}_l} d\left(x_i, \mathbf{m}_j^{(t)}\right)$$

Notice that we need to make assumptions about the dissimilarity $d(\cdot,\cdot)$ used to cluster the points in order to calculate $\nabla_{\mathbf{m}_i} d(x_0, \mathbf{m}_j)$. We take the special case where $d(\cdot,\cdot)$ is the squared euclidean norm, which is the most important for the *soft K-Means*:

$$d(x, y) = \|x - y\|_{L^2}^2$$

$$\therefore \quad \nabla_{\mathbf{m}_l} d(x_i, \mathbf{m}_j^{(t)}) = -2\delta_{lj}(x_i - \mathbf{m}_j^{(t)})$$

And we get

$$0 = \beta\ \sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij}\ 2\delta_{lj}(x_i - \mathbf{m}_j^{(t)}) = 2\beta\ \sum_{i=1}^{n} q_{il}\ (x_i - \mathbf{m}_l^{(t)})$$

$$\Rightarrow \quad \mathbf{m}_l^{(t)} = \frac{\sum_{i=1}^{n} q_{il}\ x_i}{\sum_{i=1}^{n} q_{il}}$$

Which coincides with the update step for the *soft K-Means* when d is the squared euclidean norm, according to the calculated value for q_{ij} in 3.5. We summarize these two steps:

E step: Assign $q_{ij} = \frac{\mathrm{e}^{-\beta \left\| x_i - \mathbf{m}_i^{(t-1)} \right\|_{L^2}^2}}{\sum\limits_j \mathrm{e}^{-\beta \left\| x_i - \mathbf{m}_j^{(t-1)} \right\|_{L^2}^2}}$

M step: Update the $\mathbf{m}_i$'s from $\mathbf{m}_l^{(t-1)}$ to $\mathbf{m}_l^{(t)} = \frac{\sum_{i=1}^n q_{il} \; x_i}{\sum_{i=1}^n q_{il}}$

Remark. *We proved that the soft K-Means is equivalent to the EM algorithm only for the special case when $d(x,y) = \|x-y\|_{L^2}^2$. If $d(\cdot,\cdot)$ assumes another form, it generates two completely different algorithms, for the soft K-Means, the update step continues to be*

ii. *Update the $\mathbf{m}_i$'s from $\mathbf{m}_l^{(t-1)}$ to $\mathbf{m}_l^{(t)} = \frac{\sum_{i=1}^n q_{il} \; x_i}{\sum_{i=1}^n q_{il}}$ (where $q_{il} = p_l(x_i, \; \mathbf{M}^{(t-1)}, \; D)$)*

While for the EM algorithm the update step (M step) is

ii.' (M step) *Update the $\mathbf{m}_l$'s from $\mathbf{m}_l^{(t-1)}$ to $\mathbf{m}_l^{(t)}$, the only solution of*

$$0 = \nabla_{\mathbf{m}_l} f(\beta, \mathbf{M}^{(t)}, (q_{ij})) = -\beta \, \sum_{i=1}^n \sum_{j=1}^k q_{ij} \; \nabla_{\mathbf{m}_l} d\left(x_i, \mathbf{m}_j^{(t)}\right)$$

3.6 The soft K-Means β parameter

In the previous section we showed that the soft K-Means is equivalent to the *Expectation Maximization* algorithm when $d(x,y) = \|x-y\|_{L^2}^2$, therefore, its convergence is guaranteed, according to the results in [35]. We explore a bit more the mixture model viewpoint to understand the role of the β parameter. Recall Expression 3.11 for the i^{th} model, here adapted to the squared L^2 norm:

$$\rho(x_0|\beta, \mathbf{m}_i) = \alpha \; \mathrm{e}^{-\beta \|x_0 - \mathbf{m}_i\|_{L^2}^2}$$

Noticing that $\beta \|x_0 - \mathbf{m}_i\|_{L^2}^2 = (x_0 - \mathbf{m}_i)^T \; (\frac{1}{\beta}\mathbb{I})^{-1}(x_0 - \mathbf{m}_i)$ we can rewrite the soft K-Means *pdf* as

$$\rho(x|\beta, \mathbf{M}) = \sum_{j=1}^n \frac{1}{k} \alpha \; \mathrm{e}^{-(x_0 - \mathbf{m}_i)^T \; (\frac{1}{\beta}\mathbb{I})^{-1}(x_0 - \mathbf{m}_i)}$$

Each model in the mixture has the same weight $\pi_j = \frac{1}{k}$ and is distributed as a gaussian with covariance matrix $\frac{1}{\beta}\mathbb{I}$, also, the normalization constant is $\alpha = 1/(2\pi)^{n/2}|\frac{1}{\beta}\mathbb{I}|^{1/2} = (\beta/2\pi)^{n/2}$. The covariance matrix is diagonal with eigenvalues $\frac{1}{\beta}$, threfore, the mean-squared distance from the center is $r = \frac{1}{\beta}$. This allows us an interpretation of the clusters shape geometry: they are gaussian 'blobs' centered at each $\mathbf{m}_i$ with radius $r = \frac{1}{\beta}$, all the same size (ideally, they have the same proportion of points).

The soft K-Means has more flexibility to detect clusters' round shapes and can fit better the mean-squared radius if they all share the same radius. Still we can go on trouble if the clusters are not equal sized, do not have similar mean-squared radius nor have spherical shape. If we need to capture this peculiarities, it is needed to add more parameters to our model, a task we do next section with the *Gaussian Mixture Models.*

3.7 Gaussian Mixture Models

A gaussian mixture model (GMM) is a mixture for a random variable X whose components are gaussian distributions centered at μ_i and possessing covariance matrix Σ_i, and have sizes π_i, for $i = 1, 2, \cdots k$. Hence, X is distributed according to the *pdf*

$$\rho(x) = \sum_{i=1}^{k} \pi_i \frac{\mathrm{e}^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}}{(2\pi)^{n/2}|\Sigma_i|^{1/2}} \tag{3.13}$$

Making explicit each component distribution:

$$\rho(x|\mu_i, \Sigma_i) = \frac{\mathrm{e}^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}}{(2\pi)^{n/2}|\Sigma_i|^{1/2}} \tag{3.14}$$

The parameters are $\phi = (\pi_i, \mu_i, \Sigma_i)_{i=1,2,\cdots k} \equiv (\pi, \mu, \Sigma)$ and clearly the latent variable M spans values $j \in \{1, 2, \cdots k\}$. For simplicity, we assume that the Σ's are known in advance (they won't be taken as free parameters to estimate).

For the E step, we should calculate the $Q_i(j)$'s for $j = 1, 2, \cdots k$, for each $i = 1, 2, \cdots n$ (i corresponding to the i^{th} data point), so let us we calculate the matrix of coefficients $[\ q_{ij}\ ] = [\ Q_i(j)\ ]$ similarly to what was done in Section 3.5:

$$\begin{aligned} q_{ij} &= p(M = j|x_i, \phi) = \\ &= \alpha^{-1}\ \pi_j\ \exp\Big(-\frac{1}{2}(x_i - \mu_j)^T \Sigma_j^{-1}(x_i - \mu_j)\Big) / \Big\{(2\pi)^{n/2}|\Sigma_j|^{1/2}\Big\} \end{aligned} \tag{3.15}$$

$$\text{Where } \alpha = \sum_{l=1}^{k} \pi_l\ \exp\Big(-\frac{1}{2}(x_i - \mu_l)^T \Sigma_l^{-1}(x_i - \mu_l)\Big) / \Big\{(2\pi)^{n/2}|\Sigma_l|^{1/2}\Big\}$$

Next in the M-step, we should maximize the expression

$$\begin{aligned} &\sum_{i=1}^{n}\sum_{z=1}^{k} Q_i(z) \ln \frac{p(x_i, z|\phi)}{Q_i(z)} = \sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij} \ln \Big\{\frac{p(M = j|\phi)p(x_i|M = j, \phi)}{q_{ij}}\Big\} = \\ &= \sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij} \ln \Big\{\pi_j (2\pi)^{-n/2}|\Sigma_j|^{-1/2}\exp\Big(-\frac{1}{2}(x_i - \mu_j)^T \Sigma_j^{-1}(x_i - \mu_j)\Big)/q_{ij}\Big\} = \\ &= \Big\{\sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij} \ln \pi_j - \frac{q_{ij}}{2}(x_i - \mu_j)^T \Sigma_j^{-1}(x_i - \mu_j) - \frac{q_{ij}}{2} \ln |\Sigma_j|\Big\} + constant \equiv \\ &\equiv f(\pi, \mu, \Sigma, (q_{ij})) + constant \end{aligned} \tag{3.16}$$

Equating to zero the derivatives of this final expression we obtain

$$0 = \nabla_{\mu_l} f\,(\pi, \mu, \Sigma, (q_{ij})) = -\sum_{i=1}^{n}\sum_{j=1}^{k} \frac{q_{ij}}{2}\ \nabla_{\mu_l}\{\ (x_i - \mu_j)^T \Sigma_j^{-1} (x_i - \mu_j)\ \} =$$

$$= \sum_{i=1}^{n} \frac{q_{il}}{2} \nabla_{\mu_l} \left(2\mu_l^T \Sigma_l^{-1} x_i - \mu_l^T \Sigma_l^{-1} \mu_l\right) = \sum_{i=1}^{n} q_{il} (\Sigma_l^{-1} x_i - \Sigma_l^{-1} \mu_l) =$$

$$= \sum_{i=1}^{n} q_{il} \Sigma_l^{-1} (x_i - \mu_l)$$

Considering that Σ_l^{-1} is invertible and self-adjoint, we have that

$$0 = \sum_{i=1}^{n} q_{il} \Sigma_l^{-1} (x_i - \mu_l) = \Sigma_l^{-1} \left\{ \sum_{i=1}^{n} q_{il} (x_i - \mu_l) \right\} \quad \forall\ l \ \Leftrightarrow$$

$$\Leftrightarrow\ 0 = \sum_{i=1}^{n} q_{il} (x_i - \mu_l)\ , \quad \forall\ l \quad \Leftrightarrow \tag{3.17}$$

$$\Leftrightarrow\ \mu_l = \frac{\sum_{i=1}^{n} q_{il} x_i}{\sum_{i=1}^{n} q_{il}}$$

Now we look for the optimal value for π. We look simply for the terms in $f(\pi, \mu, \Sigma, (q_{ij}))$ that depend on the π_j's. Fortunately, the parameters are not mixed together in the function f, therefore we deal only with the optimization of its term $\sum_{i=1}^{n} \sum_{j=1}^{k} q_{ij} \ln \pi_j$. So we are looking for

$$\pi^{(t)} = \underset{\sum \pi_j = 1}{\operatorname{argmax}} \{\ f(\pi, \mu, \Sigma, (q_{ij}))\ \} = \underset{\sum \pi_j = 1}{\operatorname{argmax}} \left\{ \sum_{i=1}^{n} \sum_{j=1}^{k} q_{ij} \ln \pi_j \right\}$$

For this problem, as indicated, we should perform a constraint optimization over the constraint $\sum_{i=1}^{k} \pi_i = 1$. To deal with this constraint we construct the lagrangian

$$\mathcal{L}(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{k} q_{ij} \ln \pi_j + \lambda \left(\sum_{j=1}^{k} \pi_j - 1 \right)$$

We derive it and set it equal to zero

$$0 = \frac{\partial \mathcal{L}}{\partial \pi_j}(\pi) = \sum_{i=1}^{n} \frac{q_{ij}}{\pi_j} + \lambda = \frac{\sum_{i=1}^{n} q_{ij}}{\pi_j} + \lambda$$

$$\Rightarrow \pi_j = -\left(\sum_{i=1}^{n} q_{ij}\right)/\lambda$$

And we apply the constraint for calculation of λ:

$$\begin{aligned} 1 = \sum_{j=1}^{k} \pi_j = \sum_{j=1}^{k}\sum_{i=1}^{n} q_{ij}/(-\lambda) &= \left(\sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij}\right)/(-\lambda) \\ \therefore \quad -\lambda = \sum_{i=1}^{n}\sum_{j=1}^{k} q_{ij} &= \sum_{i=1}^{n} 1 = n \\ \Rightarrow \pi_j &= \frac{1}{n}\sum_{i=1}^{n} q_{ij} \end{aligned} \tag{3.18}$$

If we took the optimization over Σ as well, the calculations would become more involved but the final result would give the empirical covariance as the estimate. We can summarize the *EM* steps adapted to this particular model as:

E step: assign $q_{ij} = \alpha^{-1}\sum_{l=1}^{k} \pi_l \exp\left(-\frac{1}{2}\left(x_i - \mu_l^{(t-1)}\right)^T \Sigma_l^{-1}\left(x_i - \mu_l^{(t-1)}\right)\right)/\left\{(2\pi)^{n/2}|\Sigma_l|^{1/2}\right\}$

where $\alpha = \sum_{l=1}^{k} \pi_l \exp\left(-\frac{1}{2}\left(x_i - \mu_l^{(t-1)}\right)^T \Sigma_l^{-1}\left(x_i - \mu_l^{(t-1)}\right)\right)/\left\{(2\pi)^{n/2}|\Sigma_l|^{1/2}\right\}$

M step: update π_j and μ_j from $\pi_j^{(t-1)}, \mu_j^{(t-1)}$ to $\pi_j^{(t)} = \frac{1}{n}\sum_{i=1}^{n} q_{ij}$ and $\mu_j^{(t)} = \sum_{i=1}^{n} q_{ij}x_i / \sum_{i=1}^{n} q_{ij}$

Below we apply the Gaussian Mixture Model to detect three clusters, plotting the calculated covariances through the ellipsoids $S_i = \{y \mid y = \Sigma_i\ x,\ \|x\| = 1\}$.

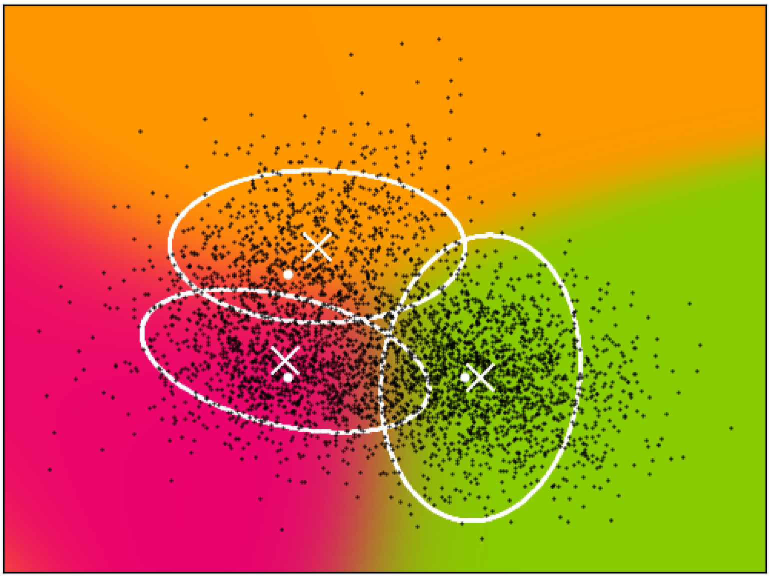


Figure 3.1: Ellipses are displayed together with the estimated centroids ('X' white marks) and the original centers (white dotted.)

3.7.1 K-Means as a Gaussian Mixture Model limit

Now we are able to prove that we can recover the hard K-Means algorithm from a *GMM* with all its Σ's equal to $\epsilon\mathbb{I}$ in the limit of $\epsilon \to 0$. Hence, assume we have a Gaussian Mixture whose components are modeled by

$$\rho(x|\mu_j, \epsilon) = \frac{\exp(-\frac{1}{2}(x_i - \mu_j)^T(\epsilon\mathbb{I})^{-1}(x_i - \mu_j))}{(2\pi)^{n/2}|\epsilon\mathbb{I}|^{1/2}}$$

Now,

$$(2\pi)^{n/2}|\epsilon\mathbb{I}|^{1/2} = (2\pi\epsilon)^{-n/2}$$

and

$$(x_i - \mu_j)^T(\epsilon\mathbb{I})^{-1}(x_i - \mu_j) = \frac{1}{2\epsilon}\|x_i - \mu_l\|^2$$

Thus Equation 3.15 simplifies to

$$q_{ij}(\epsilon) = p(M = j|x_i, \phi) =$$

$$= \left\{ \pi_j \frac{\exp(-\frac{1}{2}(x_i - \mu_j)^T(\epsilon\mathbb{I})^{-1}(x_i - \mu_j))}{(2\pi)^{n/2}|\epsilon\mathbb{I}|^{1/2}} \right\} / \left\{ \sum_{l=1}^{k} \pi_l \frac{\exp(-\frac{1}{2}(x_i - \mu_l)^T(\epsilon\mathbb{I})^{-1}(x_i - \mu_l))}{(2\pi)^{n/2}|\epsilon\mathbb{I}|^{1/2}} \right\} =$$

$$= \left\{ \pi_j \exp(-\frac{1}{2\epsilon}\|x_i - \mu_j\|^2)(2\pi\epsilon)^{-n/2} \right\} / \left\{ \sum_{l=1}^{k} \pi_l \exp(-\frac{1}{2\epsilon}\|x_i - \mu_l\|^2)(2\pi\epsilon)^{-n/2} \right\} \tag{3.19}$$

Now we prove that in the limit where $\epsilon \to 0$, $q_{ij}(\epsilon) \to r_{ij}$, where r_{ij} is the hard K-Means *responsibility function.*

Let us write

$$\delta_j \equiv \|x_i - \mu_j\|^2$$
$$\delta_{min} = \min_j \ \|x_i - \mu_j\|^2$$

So we get

$$q_{ij}(\epsilon) = \left\{ \pi_j \ \exp\left(-\frac{1}{2\epsilon}\delta_j\right) \exp\left(\frac{1}{2\epsilon}\delta_{min}\right) \right\} \bigg/ \left\{ \left(\sum_{l=1}^{k} \pi_l \ \exp\left(-\frac{1}{2\epsilon}\delta_l\right) \right) \exp\left(\frac{1}{2\epsilon}\delta_{min}\right) \right\} =$$
$$= \pi_j \ \exp\left(\frac{\delta_{min} - \delta_j}{2\epsilon}\right) \bigg/ \left\{ \sum_{l=1}^{k} \pi_l \ \exp\left(\frac{\delta_{min} - \delta_l}{2\epsilon}\right) \right\} \tag{3.20}$$

Now, consider $l^* = \underset{l}{\operatorname{argmin}} \ \delta_l$. Clearly, $\delta_{l^*} = \delta_{min}$, hence

$$\sum_{l=1}^{k} \pi_l \ \exp\left(\frac{\delta_{min} - \delta_l}{2\epsilon}\right) \geq \pi_{l^*} \ \exp\left(\frac{\delta_{min} - \delta_{l^*}}{2\epsilon}\right) = \pi_{l^*}$$

$$\Rightarrow \pi_j \ \exp\left(\frac{\delta_{min} - \delta_j}{2\epsilon}\right) \bigg/ \left\{ \sum_{l=1}^{k} \pi_l \ \exp\left(\frac{\delta_{min} - \delta_l}{2\epsilon}\right) \right\} \leq \tag{3.21}$$

$$\leq \pi_j \ \exp\left(\frac{\delta_{min} - \delta_j}{2\epsilon}\right) / \pi_{l^*}$$

So, for $j \neq l^*$, we use 3.21 in 3.20, gaining

$$q_{ij}(\epsilon) \leq \frac{\pi_j}{\pi_{l^*}} \exp\left(\frac{\delta_{min} - \delta_j}{2\epsilon}\right) \equiv c \ \exp(-\Delta/\epsilon) \tag{3.22}$$

Where $\Delta := \frac{\delta_{min}-\delta_j}{2} > 0$ for $j \neq l^*$ and $c := \frac{\pi_j}{\pi_{l^*}}$.
We notice that

$$\lim_{\epsilon \to 0} c \, \exp(-\Delta/\epsilon) = 0$$

For the case $j = l^*$, $\exp(\frac{\delta_{min}-\delta_{l^*}}{2\epsilon}) = 1$, $\forall\ \epsilon > 0$, thus we have

$$q_{il^*}(\epsilon) = \pi_{l^*} \ \exp\left(\frac{\delta_{min} - \delta_{l^*}}{2\epsilon}\right) \Big/ \left\{\sum_{l=1}^{k} \pi_l \ \exp\left(\frac{\delta_{min} - \delta_l}{2\epsilon}\right)\right\} =$$

$$= \frac{\pi_{l^*}}{\pi_{l^*} + \sum\limits_{l \neq l^*} \pi_l \exp(\frac{\delta_{min}-\delta_l}{2\epsilon})} = \tag{3.23}$$

$$= \frac{1}{1 + \sum\limits_{l \neq l^*} \frac{\pi_l}{\pi_{l^*}} \exp(\frac{\delta_{min}-\delta_l}{2\epsilon})} \xrightarrow{\epsilon \to 0} 1$$

Because

$$\lim_{\epsilon \to 0} \left\{ \sum_{l \neq l^*} \frac{\pi_l}{\pi_{l^*}} \exp\left(\frac{\delta_{min} - \delta_l}{2\epsilon}\right) \right\} = 0$$

when $\delta_{min} - \delta_l < 0$ $(i.e.\ \forall\ \ l \neq l^*\ )$.
Therefore, we see that the limit of $q_{ij}(\epsilon)$ satisfies

$$\lim_{\epsilon \to 0} \ q_{ij}(\epsilon) = \begin{cases} 1 & \text{if } \ j = \underset{l}{\text{argmin}} \ \|x_i - \mu_l\|^2 \\ 0 & \qquad\quad \text{otherwise} \end{cases}$$

As we announced this is exactly the hard assignment of data points x_i to their closest centers μ_l we saw in the K-Means algorithm. Thus, in the limit where ϵ goes to zero, the EM algorithm coincides with the Hard K-Means. Notice that when ϵ gets small, the covariance matrices of the gaussian mixture, which are all equal, make each distribution sharpen around the centers, and in the limit, this mixture becomes a weighted sum of Dirac Pics centered at the μ_l's.
The above result has an intuitive interpretation: imagine the covariances $\Sigma_i = \epsilon \ \mathbb{I}$ assume an extremely small value for ϵ. Hence, the distribution of points become a gaussian very concentrated around its center. Now suppose a data point $x \in D$ lies between two clusters' centroids. Very likely the distances from x to the centroids differ several orders of magnetude in comparison to ϵ (assuming it is really really small), so that even tiny differences in the distances make the probability of x to belong to one cluster instead of the other become extremely big, specially because the probability decays proportionaly to $\mathrm{e}^{-d^2/\epsilon}$.

3.7.2 A Fatal Flaw of GMM Clustering

The Gaussian Mixture Model allows us to fit nicely some simple data sets, because it gives us too many parameters to work on. Consider in particular the case where we update not only the π's and the μ's but also all the covariance components $(\Sigma_{ij})_{i,j=1,\cdots n}$, then we will have $m = k + dk + kd^2$ parameters to fit the data if they are all independent. It happens that data points are normally generated by a set of distributions whose total number of parameters do not exceed a number linear in both d and k (assuming k is a good candidate for the total number of clusters), so that the squared term d^2 contributes to the *overfitting* of data discussed in Chapter 1, by making m big.

Remember that the *overfitting* of data happens when there are so many parameters that we start fitting the errors, by fitting 'perfectly' the data observed at the cost of losing predictive power for newly generated samples. Find next an insightful example of overfitting in *Supervised Learning*, from *Wikipedia* [34]. In Figure 3.2 we see the 'wiggly' line fitting very well the data division between red and blue points, but this prediction do not sound realistic (a *cross validation* could account to a measure of how bad this solution is). The model behind the green line fit must have too many parameters so that it is heavily affected by the noise in the threshold between the two regions. The model behind the black line fit instead is robust against this noise. We investigate next a similar sitation but in the context of Unsupervised Learning.

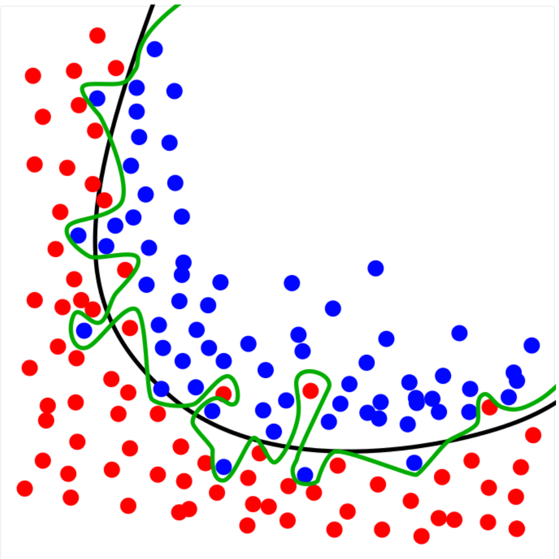


Figure 3.2: From Wikipedia - 'The green line represents an overfitted model and the black line represents a regularised model. While the green line best follows the training data, it is too dependent on it and it is likely to have a higher error rate on new unseen data, compared to the black line.'

To show how the GMM overfitting of data could affect the detection of clusters, consider the two clustered data sets in Figure 3.3 differing by only a small set of points which constitute a *noise*. The noise is small enough to be considered as an *outlier* observation, meaning that it is

not expected this noise to be considered as a whole new cluster. Then one might expect from a good clustering algorithm to detect it as part of a bigger cluster nearby. We verify instead that the GMM clustering classifies the noisy points as a whole 'new' cluster, while forces two big important clusters to merge. That is an anomally typical of *overfitting* estimations, where the parameters start to fit erros. This is an issue of the Gaussian Mixture model not fully known. Some explanations of it for special cases can be checked in [21] and [16]
Evidence shows that when one can place a gaussian distribution with very small radius around the noisy points, then the likelihood normally goes huge. This is a fatal flaw for the Gaussian Mixture Model, because when the dimension grows we verify the so-called *Curse of Dimensionality*: the sampled points become sparse, in such a way that it becomes increasingly difficult to distinguish what is an *outlier* (a noise) from what is not, because more or less every point seems to be isolated. The discussion about this curious (and awkward) phenomenon is postponed to Chapter 5, where we present some important examples and some special results of *Concentration of Measure* that make clear the concept and how to deal with (or not).

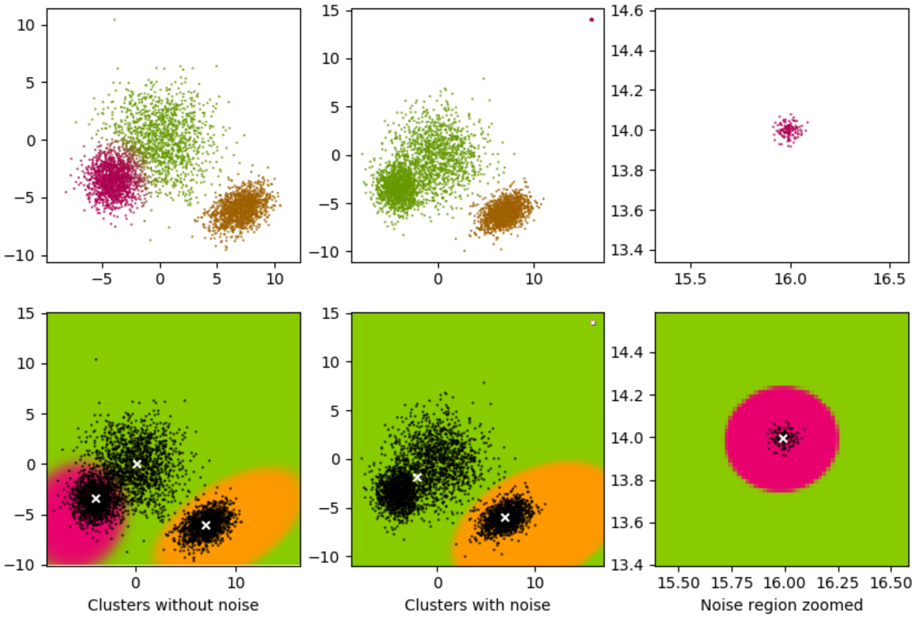


Figure 3.3: Here $K = 3$. On the left three clusters are represented together with the GMM Voronoi-like sets. In the middle a noise is added at coordinates $(16, 14)$, therefore very far from the original centroids, with radius ~ 0.001. This noise is what one would call an *outlier*, because there are very few points to justify classification of it as a whole new cluster. On the right we see the zoomed noise. Notice that because $k = 3$, the two clusters on the left are merged, and the small noise is taken as a whole cluster

Although the Expectation Maximization seems to be a nice clustering algorithm, it will not be effective against isolated noisy points that naturally occurs high dimension. We will need robust methods that can perform clustering despite the imense amount of apparent noise resulting from the intrinsic high-dimensional data sparsity. Again in Chapter 5 we reach this goal presenting an algorithm proposed by Charu Aggarwal in [2] named *ORCLUS* whose underlying idea is to project each cluster into the relevant directions, thus reducing locally in the data the dimension.

Before addressing the discussion on high dimensional data, we break and take a step backwards to a more technical chapter on the proof of K-Means convergence.

Chapter 4

A detailed study of K-Means

4.1 Clustering as an optimization program

As we have seen earlier, the K-Means clustering method produces a sequence of centroids by iterating two steps at the time, namely, the *assignment step* and the centroid *update step*, much like in the EM steps. These steps reduce the 'energy' or 'cost function' $E = E(\mathbf{M})$, in terms of (new) optimal Cluster's centers.
In this chapter we prove in detail the convergence of the K-Means algorithm exploring the idea of minimizing a cost-function in a *Constrained Optimization* framework. A brief introduction to this framework is presented in the appendix. We will restate the Clustering methods as an optimization program, and solve it following the article entitled 'K-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality' (1984) by Ismael and Selim [32]. The author of this thesis points out his contribution in making clear the referred article which has some obscure steps.
This chapter is very technical and is only concerned in delve into the details of the convergence properties of the K-Means algorithm. Despite its apparent simplicity, the K-Means *hardness* entails rich mathematical techniques, specially because it envolves the minimization of a biconvex function on a discrete domain. It is worth noting that although the K-Means first appearence dates back to *1957* in the work of Loyd [19] and popularized through the work of Macqueen [22] in *1967*, only in *1984* Ismael and Selim gave a complete proof of convergence and characterization of local optimality of the K-Means [32] and still nowadays the algorithm and its variants poses many open questions to the mathematical community (see [27]). One of the thesis author's goal with this work was to provide a rigorous, clear and self-contained source of material about the mathematics surrounding the K-Means, running away from traditional approaches which avoid hard though apparent simple details or neglect the mathematical difficulties - with a consequent loss in the richness of Cluster Analysis theory.

4.2 The special case for the K-Means

We have seen in chapter 2, through equations 2.2 to 2.3, that K-Means is an algorithm that aims to reduce the Energy function

$$E(\mathbf{M}) = \sum_{\mathbf{x}\in D} \|\mathbf{x} - \mathbf{m}_{Y(\mathbf{x})}\|_{L^2}^2$$

Remember the definition of the responsibility function, in Equation 2.4

$$r_{\mathbf{x}}^{(n)}(i) = \delta(i, Y(\mathbf{x})) = \begin{cases} 1, & \text{if} \quad \underset{j\in\{1,2,\cdots k\}}{\operatorname{argmin}} \quad d(\mathbf{x}, \mathbf{m}_j^{(n)}) = i \\ 0, & \text{if} \quad \underset{j\in\{1,2,\cdots k\}}{\operatorname{argmin}} \quad d(\mathbf{x}, \mathbf{m}_j^{(n)}) \neq i \end{cases}$$

as we have seen we can rewrite the energy function as

$$E(\mathbf{M}) = \sum_{\mathbf{x}\in D} \sum_{i=1}^{k} r_{\mathbf{x}}^{(n)}(i) \ \|\mathbf{x} - \mathbf{m}_i\|_{L^2}^2$$

which suggests a generalization to the special cases of Soft K-Means:

$$E(\mathbf{M}) = \sum_{\mathbf{x}\in D} \sum_{i=1}^{k} p_i(\mathbf{x}, \mathbf{M}, D) \ \|\mathbf{x} - \mathbf{m}_i\|_{L^2}^2$$

Different modelings for the shape of $p_i(\mathbf{x}, \mathbf{M}, D)$ lead to new clustering methods, for example, the soft K-Means algorithm with $p_i(\mathbf{x}, \mathbf{M}, D) \ \propto \ \mathrm{e}^{-\beta d(\mathbf{x}, \ \mathbf{m}_i)}$. In this section we are going to study the clustering procedures associated to a family of Energy functions which give rise to the hard K-Means algorithms.
In the above formula there are two special features: the probabilities assignments for each $\mathbf{x} \in D$ and a notion of distance of $\mathbf{x}$ from the centers $\mathbf{m}_i$, which can be generically thought of a metric evaluation $d(\mathbf{x}, \mathbf{m}_i)$. Denoting the elements of D by $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n$ we could define the matrix of probabilities assignments as

$$\mathbf{P} = [\ p_{ij} \ ] := [\ p_i(\mathbf{x}_j, \mathbf{M}, D) \ ] \in \mathcal{M}_{k\times n}(\mathbb{R})$$

A first generalization would be thinking that the p_{ij}'s have no functional dependency on $\mathbf{M}$ and D, meaning that they are real variables subject to the constraints

$$\sum_{i=1}^{k} p_{ij} = 1 \text{ for } j = 1, 2, \cdots n$$

And for the distances between $\mathbf{x}$'s and the centers $\mathbf{m}_i$, we could define a matrix of values $d(\mathbf{x}_j, \mathbf{m}_i)$. However, instead of considering any metric, we use the notion of a *dissimilarity measure*, which is a function of

$$\begin{aligned} D\colon \mathcal{X} \times \mathcal{X} &\to \mathbb{R} \\ (x,y) &\mapsto D(x,y) \end{aligned}$$

satisfying the following properties:

i. $D(x, y) \geq 0$;

ii. $D(x, y) = D(y, x)$;

iii. $D(x, y) = 0 \Leftrightarrow x = y$.

It is desirable, but not necessarily true, that the *dissimilarity measures* satisfy

iv. $D(x, z) + D(z, y) \geq D(x, y)$; for every $x, y, z \in \mathcal{X}$
When the last condition is satisfied D is a metric for the subset $\mathcal{X} \subset \mathbb{R}^{\mathrm{d}}$.

As our primary focus is the euclidean distance, $D(\cdot,\cdot)$ in all facts is a metric $d(\cdot,\cdot)$. However, often in real computational problems one can solely rely on the data points $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n$, and at most combinations of them like the clusters means $\mathbf{m}_i$. It might be the case that the notion of distance works only for the restricted sets of the state space $\mathbb{R}^{\mathrm{d}}$. We warn the reader that we are using the same notation D both for the dissimilarity measure and for indicating the set of data points $D = \{\mathbf{x}_1, \cdots \mathbf{x}_n\}$. The symbol D was chosen to maintain resemblance to $d(\cdot,\cdot)$, apart from being the first letter of 'Dissimilarity'. We are confident that identification of which D we are considering is easy in context.
Now, established the notion of *dissimilarity*, let us define the matrix

$$\mathbf{D} = \left[\ D_{ij}\ \right] := \left[\ D(\mathbf{x}_i, \mathbf{m}_j)\ \right] \in \mathcal{M}_{n\times k}(\mathbb{R})$$

Given this, we extend our energy function to the family of *objective functions*

$$f(\mathbf{P}, \mathbf{M}) = \sum_{i=1}^{n} \sum_{j=1}^{k} p_{ji}\ D(\mathbf{x}_i, \mathbf{m}_j)$$

Notice that the previous can be cast in a matrix notation as $f(\mathbf{P}, \mathbf{M}) = \mathrm{tr}\,(\mathbf{P}^T\mathbf{D})$. This dissimilarity matrix can be seen as a function $\mathbf{D} = \mathbf{D}(\mathbf{M})$. Now, we cast our clustering problem as the program

$$\begin{cases} \text{minimize} & f(\mathbf{P}, \mathbf{M}) \\ \text{subject to} & \displaystyle\sum_{j=1}^{k} p_{ij} = 1\ , \qquad i = 1, 2, \cdots n \\ & p_{ij} = 0 \text{ or } 1\ , \quad i = 1, 2, \cdots n \\ & \qquad\qquad\qquad\quad j = 1, 2, \cdots k \end{cases} \tag{4.1}$$

For the special case in which $D(\mathbf{x}_i, \mathbf{m}_j) = \|\mathbf{x}_i - \mathbf{m}_j\|^2_{L^2}$, the objective function of 4.1 becomes biconvex. But in a general framework, the minimization algorithm will work in two successive steps: minimization of $\mathbf{P}$ given a fixed $\mathbf{M}$, then minimization in $\mathbf{M}$.

At this point, we give a proof for the convergence of K-Means using the techniques depicted in the reference [32].
We need first to understand how to handle the constraint $p_{ij} \in \{0,1\}$, as it cannot be put in the form $g(\mathbf{P},\mathbf{M}) = 0$ with g differentiable. Defining the set

$$S = \{\mathbf{P} \in \mathcal{M}_{k\times n}(\mathbb{R}) \mid \sum_{i=1}^{k} p_{ij} = 1 \; \forall \; j = 1, 2, \cdots n\}$$

and then transforming the original problem to an equivalent one, we will check that for the former, its solutions should necessarily lie in the extremes of S, which are composed of 0's and 1's.
We begin proving the

Theorem 4.2.1. *The extreme points of S under the supremum norm for matrices satisfy the constraints of the problem 4.1*

Proof. Definition $\mathbf{P}$ is an extreme point of the convex set S if it doesn't lie in any open segment joining two points of S.
Suppose its $j-th$ column does not lie in the canonical basis. Then it should have at least two positive non-zero components, say $p_{i_1 j} = \alpha$ and $p_{1_2 j} = \beta$.
Defining $\Delta = (\alpha+\beta)/2$, one can construct two new matrices, $\mathbf{P}_1$ and $\mathbf{P}_2$, differing of $\mathbf{P}$, respectively, only by the components $p1_{i_1 j} = \alpha{+}\Delta$, $p1_{i_2 j} = \beta{-}\Delta$ and $p2_{i_1 j} = \alpha{-}\Delta$, $p2_{i_2 j} = \beta{+}\Delta$.
By this construction, is easy to see that $\mathbf{P}$ should lie in the line between $\mathbf{P}_1$ and $\mathbf{P}_2$, whuch contradicts the extreme point hypothesis.
Therefore, each of its columns should be a vector of 0's or 1's, adding up to 1. □

Next we define a new related problem and prove its equivalence to the original one. In order to that, we need the

Definition 4.2.1. *The reduction function associated to problem 4.1 is*

$$F(\mathbf{P}) = \min\{f(\mathbf{P},\mathbf{M}) \mid \mathbf{M} \in \mathcal{M}_{d\times k}(\mathbb{R})\}$$

Lemma 4.2.2. *The reduced function F is concave.*

Proof. Consider two arbitrary points $\mathbf{P}_1$ and $\mathbf{P}_2$, and a scalar $t \in [0,1]$. As $f(\mathbf{P},\mathbf{M}) = \mathrm{tr}\,(\mathbf{P}^T\mathbf{D})$ (where $\mathbf{D} = \mathbf{D}(\mathbf{M})$), the linearity of the trace implies

$$\begin{aligned} f(\ (1-t)\mathbf{P}_1 + t\mathbf{P}_2, \mathbf{M}\ ) &= \mathrm{tr}\,(\ ((1-t)\mathbf{P}_1 + t\mathbf{P}_2)^T\mathbf{D}\ ) = \\ &= (1-t)\,\mathrm{tr}\,(\mathbf{P}_1^T\mathbf{D}) + t\,\mathrm{tr}\,(\mathbf{P}_2^T\mathbf{D}) = \\ &= (1-t)f(\mathbf{P}_1,\mathbf{M}) + tf(\mathbf{P}_2,\mathbf{M}) \end{aligned}$$

Then

$$\begin{aligned} F(\ (1-t)\mathbf{P}_1 + t\mathbf{P}_2\ ) &= \min\{f((1-t)\mathbf{P}_1 + t\mathbf{P}_2, \mathbf{M}) \mid \mathbf{M} \in \mathcal{M}_{d\times k}(\mathbb{R})\} = \\ &= \min_{\mathbf{M}}\{(1-t)f(\mathbf{P}_1,\mathbf{M}) + tf(\mathbf{P}_2,\mathbf{M})\} \geq \\ &\geq (1-t)\ \min_{\mathbf{M}}\{f(\mathbf{P}_1,\mathbf{M})\} + t\ \min_{\mathbf{M}}\{f(\mathbf{P}_2,\mathbf{M}) = \\ &= (1-t)\ F(\mathbf{P}_1) + t\ F(\mathbf{P}_2) \end{aligned}$$

(Notice there was a change in notation from 'min$\{\ \cdot \mid \mathbf{M} \in \mathcal{M}_{d\times k}(\mathbb{R})\}$' to the cleaner form '$\min_{\mathbf{M}}\{\ \cdot\ \}$')
Hence F is concave. □

Finally, we have the following definition and theorem:

Definition 4.2.2. *The Reduced Problem for 4.1 is the optimization program:*

$$\begin{cases} \textit{minimize} & F(\mathbf{P}) \\ \textit{subject to} & \mathbf{P} \in S \end{cases} \tag{4.2}$$

Lemma 4.2.3. *Problems 4.1 and 4.2 are equivalent.*

Proof. Suppose there exists a solution $\mathbf{P}_0$ for problem 4.2. Then it should be an extreme point of S or it can be displaced to another solution $\mathbf{P}_1$ which has this property, if not, there would exist two other points $\mathbf{P}_1$ and $\mathbf{P}_2$ such that $\mathbf{P}_0 = (1-t)\mathbf{P}_1 + t\mathbf{P}_2$ for some $t \in (0,1)$.
If $F(\mathbf{P}_1) < F(\mathbf{P}_2)$, then $F(\mathbf{P}_1) < F(\mathbf{P}_1) + t(F(\mathbf{P}_2) - F(\mathbf{P}_1))$, hence, by the concavity of F,

$$F(\mathbf{P}_0) \leq F(\mathbf{P}_1) < (1-t)F(\mathbf{P}_1) + tF(\mathbf{P}_2) \leq F(\ (1-t)\mathbf{P}_1 + t\mathbf{P}_2\ ) = F(\mathbf{P}_0)$$

If $F(\mathbf{P}_1) > F(\mathbf{P}_2)$, the proof goes all the same of the last part.
For the case in which $F(\mathbf{P}_1) = F(\mathbf{P}_2)$,

$$(\lambda - 1)F(\mathbf{P}_1) + \lambda F(\mathbf{P}_2) = F(\mathbf{P}_1) = F(\mathbf{P}_2)\ \forall\ \lambda \in (0,1)$$

Then

$$F(\mathbf{P}_1) = (t-1)F(\mathbf{P}_1) + tF(\mathbf{P}_2) \leq F(\ (1-t)\mathbf{P}_1 + t\mathbf{P}_2\ ) = F(\mathbf{P}_0) \leq F(\mathbf{P}_1)$$

Hence $F(\mathbf{P}_0) = F(\mathbf{P}_1)$. A similar reasoning applies to $F(\mathbf{P}_2)$, and in fact we could prove that applies for the whole line between $\mathbf{P}_1$ and $\mathbf{P}_2$. Now, if one of these endpoints are extreme points of S, we're done, else, consider the set

$$A = \{t \geq 0 \mid \mathbf{P}_1 + t(\mathbf{P}_2 - \mathbf{P}_1) \in S\}$$

The condition $\sum_{i=1}^{k} p_{ij} = 1$ implies that S is bounded in whatever matrix norm we consider, so should A be bounded as well. Also, it is easy to see that S is closed. Hence the supremum $\alpha = \sup A$ exists and $\hat{\mathbf{P}}_0 := \mathbf{P}_1 + \alpha(\mathbf{P}_2 - \mathbf{P}_1) \in S$.

By repeating the arguments above, it is easy to show that $F(\hat{\mathbf{P}}_0) = F(\mathbf{P}_0)$. Finally, $\hat{\mathbf{P}}_0$ should be an extreme point, else, one could find a $t > \alpha$ such that $\mathbf{P}_1 + t(\mathbf{P}_2 - \mathbf{P}_1)$, a contradiction. Therefore, we can displace $\mathbf{P}_0$ to another minimum $\hat{\mathbf{P}}_0$ (connected by a straight which keeps F constant) which is an extreme point of S.

Now, given such an extreme point $\mathbf{P}_0 \in S$, as it is a solution for problem 4.2, it satisfies

$$\begin{aligned} F(\mathbf{P}_0) \leq \min\{f(\mathbf{P},\mathbf{M}) \mid \mathbf{M} \in \mathcal{M}_{k\times d}(\mathbb{R})\} &\leq f(\mathbf{P},\mathbf{M}) \\ \forall \;\; \mathbf{P} \in S \text{ and } \mathbf{M} &\in \mathcal{M}_{k\times d}(\mathbb{R}) \end{aligned}$$

$$F(\mathbf{P}_0) = \min\{f(\mathbf{P}_0,\mathbf{M}) \mid \mathbf{M} \in \mathcal{M}_{k\times d}(\mathbb{R})\} = f(\mathbf{P}_0,\mathbf{M}_{min})$$

Where $\mathbf{M}_{min} \in \mathcal{M}_{k\times d}(\mathbb{R})$ is such that $f(\mathbf{P}_0,\mathbf{M}_{min}) \leq f(\mathbf{P}_0,\mathbf{M})$ for all $\mathbf{M}$. Therefore, $(\mathbf{P}_0,\mathbf{M}_{min})$ is a solution for problem 4.1 as $\mathbf{P}_0$ is a matrix of 0's and 1's with rows summing up to 1 and $f(\mathbf{P}_0,\mathbf{M}_{min}) \leq f(\mathbf{P},\mathbf{M})$ for all $\mathbf{P}$ and $\mathbf{M}$.

Clearly, we should be able to find an explicit expression for $\mathbf{M}_{min}$ while computing the minima for $\mathbf{P}_0$. The proposed algorithm (next) will naturally do that, through iteration of two steps.

Now, the converse. If $(\mathbf{P}_0,\mathbf{M}_0)$ is a solution for problem 4.2, then $f(\mathbf{P}_0,\mathbf{M}_0) \leq f(\mathbf{P},\mathbf{M})$ for all $\mathbf{M}$ and $\mathbf{P}$. Thus, for every fixed $\mathbf{P}$,

$$\begin{aligned} & f(\mathbf{P}_0,\mathbf{M}_0) \leq f(\mathbf{P},\mathbf{M}) \;\; \forall \;\; \mathbf{M} \\ & \Rightarrow f(\mathbf{P}_0,\mathbf{M}_0) \leq \min_{\mathbf{M}} f(\mathbf{P},\mathbf{M}) = F(\mathbf{P}) \;\; \forall \;\; \mathbf{P} \end{aligned}$$

Now, $F(\mathbf{P}_0) = \min_{\mathbf{M}} f(\mathbf{P}_0,\mathbf{M}) \leq f(\mathbf{P}_0,\mathbf{M}_0) \leq F(\mathbf{P}) \;\; \forall \;\; \mathbf{P}$.

Thus, $\mathbf{P}_0$ is an extreme point of S satisfying

$$F(\mathbf{P}_0) \leq F(\mathbf{P}) \;\; \forall \;\; \mathbf{P} \in S$$

i.e., $\mathbf{P}_0$ is a solution for the problem 4.2

□

Later on, we will go back to the funtion F. Now, we continue to the problem 4.1 but taking values in S (*i.e.*, forgetting the condition $p_{ij} \in \{0,1\}$, as it is necessarily satisfied). Defining the vector of ones as $E = (1,1,\cdots 1)$, and treating the columns of the matrix $\mathbf{P}$ as n variables $(\mathbf{p}_1,\mathbf{p}_2,\cdots \mathbf{p}_n)$, we can redefine S in the following terms:

$$S = \{\mathbf{P} \equiv (\mathbf{p}_1,\mathbf{p}_2,\cdots \mathbf{p}_n) \mid \langle E,\mathbf{p}_i\rangle = 1 \;,\;\; \mathbf{p}_i \geq 0 \;;\;\; \forall \, i = 1,2,\cdots n\}$$

Because of the equivalence of the problems 4.1 and 4.2, necessarily an optimal solution should be an extreme point of S, hence automatically satisfying the condition $p_{ij} \in \{0,1\}$. Thus, we can disregard this condition in 4.1 and recast it as the program

$$\begin{cases} \text{minimize} & f(\mathbf{p}_1,\mathbf{p}_2,\cdots \mathbf{p}_n;\mathbf{M}) & \\ \text{subject to} & h_i(\mathbf{p}_i) = 0 \;, & i = 1,2,\cdots n \\ & \mathbf{g}_i(\mathbf{p}_i) \leq 0 \;, & i = 1,2,\cdots n \end{cases} \tag{4.3}$$

Where

$$\begin{cases} h_i(\mathbf{p}_i) &= \langle E, \mathbf{p}_i\rangle - 1 \qquad \Rightarrow \nabla h_i(\mathbf{p}_i) = E \\ \mathbf{g}_i(\mathbf{p}_i) &= -\mathbf{p}_i \qquad \Rightarrow \nabla_{\mathbf{p}_i}\mathbf{g}_i(\mathbf{p}_i) = -\mathbf{I} \end{cases}$$

This way, the KKT points should satisfy:

$$\begin{cases} 0 &= \nabla_{\mathbf{p}_i} f(\mathbf{P},\mathbf{M}) + \mathbf{u}_i^T\nabla_{\mathbf{p}_i}\mathbf{g}_i(\mathbf{p}_i) + v_i\nabla_{\mathbf{p}_i}h_i(\mathbf{p}_i) = \nabla_{\mathbf{p}_i} f(\mathbf{P},\mathbf{M}) - \mathbf{u}_i^T + v_iE \\ 0 &= \nabla_{\mathbf{m}_i} f(\mathbf{P},\mathbf{M}) + \mathbf{u}_i^T\nabla_{\mathbf{m}_i}\mathbf{g}_i(\mathbf{p}_i) + v_i\nabla_{\mathbf{m}_i}h_i(\mathbf{p}_i) = \nabla_{\mathbf{m}_i} f(\mathbf{P},\mathbf{M}) \\ 0 &= (\mathbf{u}_i)^j \cdot \mathbf{g}_i^j(\mathbf{p}_i) = u_i^j p_{ji} \quad (\Leftrightarrow \; \mathbf{u}_i \otimes \mathbf{g}_i = 0) \end{cases}$$

Therefore, we should look for points satisfying

i. $\langle E, \mathbf{p}_i\rangle = 1$

ii. $\nabla_{\mathbf{m}_i} f(\mathbf{P},\mathbf{M}) = 0$

iii. $\nabla_{\mathbf{p}_i} f(\mathbf{P},\mathbf{M}) + v_iE = \mathbf{u}_i^T \geq 0$

iv. $(\nabla_{\mathbf{p}_i} f(\mathbf{P},\mathbf{M}) + v_iE) \otimes \mathbf{p}_i = 0$

In the following we will define the notion of a partial optimal solution, and prove that it satisfies the KKT conditions (i)-(iv), then we are going to construct an algorithm which generates a convergent sequence to such a solution. This algorithm is the K-Means, for when $D(\mathbf{x},\mathbf{m}) = \|\mathbf{x}-\mathbf{m}\|_{L^2}^2$.

Definition 4.2.3. *A point* $(\mathbf{P}_0,\mathbf{M}_0)$ *is a partial optimal solution for problem 4.2 if*

$$f(\mathbf{P}_0,\mathbf{M}_0) \leq f(\mathbf{P},\mathbf{M}_0) \quad \forall\ \mathbf{P} \in S$$
$$\textit{and}$$
$$f(\mathbf{P}_0,\mathbf{M}_0) \leq f(\mathbf{P}_0,\mathbf{M}) \quad \forall\ \mathbf{M} \in \mathcal{M}_{d\times k}(\mathbb{R})$$

Associated to a partial optimal solution, there are the two following problems:

$$\text{Problem } P_1 : \text{given } \hat{\mathbf{M}}\text{ , minimize } f(\mathbf{P};\hat{\mathbf{M}}) \text{ subject to } \mathbf{P} \in S$$
$$\text{Problem } P_2 : \text{given } \hat{\mathbf{P}}\text{ , minimize } f(\hat{\mathbf{P}};\mathbf{M}) \text{ subject to } \mathbf{M} \in \mathcal{M}_{d\times k}(\mathbb{R})$$

For problem P_1, because $\hat{\mathbf{M}}$ is fixed, its KKT points must satisfy conditions (i), (iii) and (iv). For problem P_2, the KKT conditions reduce to only (ii). With this in mind we are ready to prove the

Theorem 4.2.4. *Given a point* $(\mathbf{P}_0,\mathbf{M}_0)$*, suppose* $f = f(\mathbf{P}_0,\mathbf{M})$ *is differentiable at* $\mathbf{M}_0$ *(which is the same as to say that* $D = D(\mathbf{x}_i,\mathbf{m})$ *is differentiable function of* $\mathbf{m}$*, for every* $\mathbf{x}_i$*). If* $(\mathbf{P}_0,\mathbf{M}_0)$ *is a partial optimal solution of problem 4.2 then it is a Karush-Kuhn-Tukker point for the same problem.*

Proof. The Karush-Kuhn-Tuker conditions for 4.2 are given by (i) to (iv). If $(\mathbf{P}_0, \mathbf{M}_0)$ is a partial optimal solution for 4.2 then it solves P_1 and P_2, simultaneously, given $\hat{\mathbf{P}} = \mathbf{P}_0$ and $\hat{\mathbf{M}} = \mathbf{M}_0$, by its very definition.
Therefore, given the fixed value $\mathbf{M}_0$, $\mathbf{P}_0$ is a minimum for problem P_1 and necessarily satisfy its KKT conditions which coincides with (i), (iii) and (iv) for $\mathbf{M} = \mathbf{M}_0$, and for fixed $\mathbf{P}_0$, $\mathbf{M}_0$ is a minimum for problem P_2, thus, together with the differentiability condition, it should satisfy the KKT conditions which in turn reduces to (ii) with $\mathbf{P} = \mathbf{P}_0$. □

Finally we can devise a generic restatement of the K-Means algorithm based on the following steps:

Algorithm 4.2.1.

i. *Select an initial point* $\mathbf{M}_0 \in \mathbb{R}^{nk}$*, solve* P_1 *for* $\mathbf{M} = \mathbf{M}_0$*. Then select an optimal solution* $\mathbf{P}_0$ *of* P_1*. Set* $r = 0$.

ii. *Define* $\mathbf{M}_{r+1}$ *as a solution of* P_2 *with* $\mathbf{P} = \mathbf{P}_r$*. If* $f(\mathbf{P}_r, \mathbf{M}_{r+1}) = f(\mathbf{P}_r, \mathbf{M}_r)$*, stop; set* $(\mathbf{P}_{min}, \mathbf{M}_{min}) = (\mathbf{P}_r, \mathbf{M}_{r+1})$*; otherwise go to step* (iii) .

iii. *Define* $\mathbf{P}_{r+1}$ *as a solution of* P_1 *with* $\mathbf{M} = \mathbf{M}_{r+1}$*. If* $f(\mathbf{P}_{r+1}, \mathbf{M}_{r+1}) = f(\mathbf{P}_r, \mathbf{M}_{r+1})$ *stop; set* $(\mathbf{P}_{min}, \mathbf{M}_{min}) = (\mathbf{P}_{r+1}, \mathbf{M}_{r+1})$*; otherwise let* $r = r + 1$ *and go to step* (ii) .

The above algorithm is almost the same as the one shown previously in Section 2.3.1 with the difference that it is designed to only continue if it detects a strict decrease of f, hence, by design, it cannot make f decrease infinitely, as proved in the

Theorem 4.2.5. *The algorithm 4.2.1 converges to a partial optimal solution of problem 4.1 in a finite number of iterations.*

Proof. First we prove that, an extreme point of the set S can be visited at most once by the algorithm before it stops, for suppose not. Then it would mean that there exists $r_1 \neq r_2$ such that $\mathbf{P}_{r_1} = \mathbf{P}_{r_2}$.
The step (ii) assures that $\mathbf{M}_{r_1+1}$ and $\mathbf{M}_{r_2+1}$ are optimal solutions of P_2 for $\mathbf{M} = \mathbf{M}_{r_1}$ and $\mathbf{M} = \mathbf{M}_{r_2}$, respectively, then

$$f(\mathbf{P}_{r_1}, \mathbf{M}_{r_1+1}) \leq f(\mathbf{P}_{r_1}, \mathbf{M}_{r_2+1}) = f(\mathbf{P}_{r_2}, \mathbf{M}_{r_2+1})$$
$$f(\mathbf{P}_{r_2}, \mathbf{M}_{r_2+1}) \leq f(\mathbf{P}_{r_2}, \mathbf{M}_{r_1+1}) = f(\mathbf{P}_{r_1}, \mathbf{M}_{r_1+1})$$

So we conclude that $f(\mathbf{P}_{r_1}, \mathbf{M}_{r_1+1}) = f(\mathbf{P}_{r_2}, \mathbf{M}_{r_2+1})$ a contradiction, as the sequence of f's is strictly decreasing.
By a similar reasoning to that in lemma 4.2.3, the solution for P_1 necessarily lies on the extreme points of S, and taking in consideration that the extreme points of S form a finite set, we conclude that a partial optimal solution should be reached in a finite number of iterations (if not, the set of extreme points of S would be infinite). □

Remark. *A solution for P_1 is straightforward, given a fixed* $\mathbf{M}$*. For instance, for each j, by setting $p_{rj} = 1$ if $D(\mathbf{x}_r, \mathbf{p}_j) \leq D(\mathbf{x}_s, \mathbf{p}_j)$ for $s = 1, 2, \cdots k$ and $p_{sj} = 0$ for $s \neq r$. It is readily seen that this is a minimum.*

So far, we have seen that partial optimal solutions are attained in the final iteration of algorithm 4.2.1. But not necessarily these points are local minima. In the following we investigate under what conditions one can guarantee that the result of the algorithm is in fact a local minimum, and apply this analysis for the special cases when $D(\cdot, \cdot)$ is the euclidean norm and in general the L^p norm.
We will need from now on to assume that $\mathbf{M}$ assumes values from a compact set V. For the K-Means, the centers are selected from convex combinations of some data points (by the very definition of center of a cluster), therefore each $\mathbf{m}_i$ take values at least in the set of all convex combinations of data points x_i, which is the *convex hull* of the data set D (the convex hull is compact).
In what follows, a series of results are proven, showing that in the end, everything will reduce to check if the set $A(\mathbf{P})$ defined below is singleton:

Definition 4.2.4. $A(\mathbf{P}) = \{\mathbf{M}_{min} \mid \mathbf{M}_{min}$ *minimizes* $f(\mathbf{P}, \mathbf{M})\ ,\ \mathbf{M} \in V\}$

Hence $\bar{\mathbf{M}} \in A(\mathbf{P})$ if and only if $f(\mathbf{P}, \bar{\mathbf{M}}) \leq f(\mathbf{P}, \mathbf{M})$ for all $\mathbf{M} \in V$.
We start stating the well established result of convex analysis, by Danskin. A general statement and proof can be found in the appendix, and more references can be seen at [18] (S. Lasdon, chapter 8, page 420). Another good reference is [29]

Theorem 4.2.6 (Danskin's)**.** *Let $f = f(\mathbf{P}, \mathbf{M})$ be defined for $\mathbf{P} \in \mathbb{R}^{nk}$ and $\mathbf{M} \in V$, where V is compact.*
Define $A(\mathbf{P})$ as above (definition 4.2.4) and the function

$$F(\mathbf{P}) = \min\{f(\mathbf{P}, \mathbf{M}) \mid \mathbf{M} \in V\}$$

Also, let the one-sided directional derivative of F at $\mathbf{P}$ in the direction $\mathbf{d}$ be given as

$$DF(\mathbf{P}; \mathbf{d}) = \lim_{\alpha \to 0^+} \frac{F(\mathbf{P} + \alpha \mathbf{d}) - F(\mathbf{P})}{\alpha}$$

If f and the partial derivatives $\partial f / \partial p_{ij}$ are continuous, then $DF(\mathbf{P}; \mathbf{d})$ exists for any $\mathbf{d}$ at any point $\mathbf{P}$ and is given by

$$DF(\mathbf{P}; \mathbf{d}) = \min\{\langle \nabla_{\mathbf{P}} f(\mathbf{P}, \mathbf{M}), \mathbf{d} \rangle \mid \mathbf{M} \in A(\mathbf{P})\}$$

Remark. *In the above,* $\langle \nabla_{\mathbf{P}} f(\mathbf{P}, \mathbf{M}), \mathbf{d} \rangle = \sum_{i,j} \partial f / \partial p_{ij}\ d_{ij}$

Now we restate the optimality conditions for problem 4.2 in terms of one-sided directional derivatives through the

Lemma 4.2.7. *Consider problem $\min\{F(\mathbf{P}) \mid \mathbf{P} \in S\}$ (4.2) with S a convex set. A point $\bar{\mathbf{P}}$ is a local minimum if and only if $DF(\mathbf{P}; \mathbf{d}) \geq 0$, for each feasible direction $\mathbf{d}$ at $\bar{\mathbf{P}}$.*

This result is quite involved and a proof is found at [30] (theorem 27.1, item (ii))
Next we characterize the local optimality condition through the

Theorem 4.2.8. *Let* $(\bar{\mathbf{P}}, \bar{\mathbf{M}})$ *be a point such that* $\bar{\mathbf{P}}$ *is an extreme of the set* S *(as defined above) and* $\bar{\mathbf{M}} \in A(\bar{\mathbf{P}})$. *Then* $\bar{\mathbf{P}}$ *is a local minimum of problem 4.2 if and only if*

$$F(\bar{\mathbf{P}}) = f(\bar{\mathbf{P}}, \bar{\mathbf{M}}) \leq \min\{f(\mathbf{P}, \mathbf{M}) \mid (\mathbf{P}, \mathbf{M}) \in S \times A(\bar{\mathbf{P}})\} \tag{4.4}$$

Proof. Assume condition 4.4 holds. For every $\hat{\mathbf{M}} \in A(\hat{\mathbf{P}})$, by definition of $A(\bar{\mathbf{P}})$ and the assumption restricted to $S \times \{\hat{\mathbf{M}}\}$,

$$f(\bar{\mathbf{P}}, \hat{\mathbf{M}}) \leq f(\bar{\mathbf{P}}, \bar{\mathbf{M}}) \leq \min\{f(\mathbf{P}, \hat{\mathbf{M}}) \mid \mathbf{P} \in S\}$$

Which implies that $\bar{\mathbf{P}}$ is a local minimum for $f(\cdot, \hat{\mathbf{M}})$,thus for any feasible direction $\mathbf{d}$,

$$\langle \nabla_{\mathbf{P}} f(\bar{\mathbf{P}}, \hat{\mathbf{M}}), \mathbf{d} \rangle \geq 0$$

This works for every $\hat{\mathbf{M}} \in A(\bar{\mathbf{P}})$, so we have

$$DF(\bar{\mathbf{P}}; \mathbf{d}) = \min\{\langle \nabla_{\mathbf{P}} f(\bar{\mathbf{P}}, \mathbf{M}), \mathbf{d} \rangle \mid \mathbf{M} \in A(\bar{\mathbf{P}})\} \geq 0$$

Hence by lemma 4.2.7, $\bar{\mathbf{P}}$ is a local minimum for problem 4.2.
Now assume $\bar{\mathbf{P}}$ is a local minimum of F. Then for any feasible direction $\mathbf{d}$, $DF(\bar{\mathbf{P}}; \mathbf{d}) \geq 0$. By Danskin's theorem, this means that

$$\langle \nabla_{\mathbf{P}} f(\bar{\mathbf{P}}, \mathbf{M}), \mathbf{d} \rangle \geq 0 \quad \text{for all } \mathbf{M} \in A(\bar{\mathbf{P}})$$

Fix a value for $\hat{\mathbf{M}} \in A(\bar{\mathbf{P}})$. The function f is linear on $\mathbf{P}$ and as S is convex, given any point $\mathbf{P} \in S$, the vector $\mathbf{d} = \mathbf{P} - \bar{\mathbf{P}}$ becomes a feasible direction for the interval $(0, 1)$. Hence we compute

$$f(\mathbf{P}, \hat{\mathbf{M}}) = f(\bar{\mathbf{P}}, \hat{\mathbf{M}}) + \langle \nabla_{\mathbf{P}} f(\bar{\mathbf{P}}, \hat{\mathbf{M}}), \mathbf{d} \rangle \geq f(\bar{\mathbf{P}}, \hat{\mathbf{M}}) \tag{4.5}$$

But by definition of $A(\bar{\mathbf{P}})$, $F(\bar{\mathbf{P}}) = f(\bar{\mathbf{P}}, \mathbf{M})$, for all $\mathbf{M} \in A(\bar{\mathbf{P}})$. So condition ii. together with the arbitrariness of $\hat{\mathbf{M}}$ gives us

$$\begin{aligned} F(\bar{\mathbf{P}}) =& f(\bar{\mathbf{P}}, \hat{\mathbf{M}}) \leq f(\mathbf{P}, \hat{\mathbf{M}}) \text{ for all } \mathbf{P} \in S \text{ and } \hat{\mathbf{M}} \in A(\bar{\mathbf{P}}) \\ \Rightarrow \ & F(\bar{\mathbf{P}}) \leq \min\{f(\mathbf{P}, \mathbf{M}) \mid (\mathbf{P}, \mathbf{M}) \in S \times A(\bar{\mathbf{P}})\} \end{aligned}$$

□

Finally we can connect the characterization of minimum points with partial optimal solutions, under the condition that $A(\bar{\mathbf{P}})$ is singleton:

Theorem 4.2.9. *If* $A(\bar{\mathbf{P}})$ *is singleton and* $(\bar{\mathbf{P}}, \bar{\mathbf{M}})$ *is a partial optimal solution of problem 4.1, then* $\bar{\mathbf{P}}$ *is a local minimum for problem 4.2*

Remark. *Ideally, it is desirable that $\bar{\mathbf{P}}$ is the global minimum of F, in this case, by the equivalence between 4.1 and 4.2, the pair $\bar{\mathbf{P}}$ and $\bar{\mathbf{M}} \in A(\bar{\mathbf{P}})$ would be the global minimum for f. The previous result is the best we can do, and a possible improvement of the K-Means algorithm would be to run it many times and get the result that makes f smaller.*

Proof. If $(\bar{\mathbf{P}}, \bar{\mathbf{M}})$ is a partial optimal solution, then

$$f(\bar{\mathbf{P}}, \bar{\mathbf{M}}) \leq \min\{f(\mathbf{P}, \bar{\mathbf{M}}) \mid \mathbf{P} \in S\} = \min\{f(\mathbf{P}, \bar{\mathbf{M}}) \mid (\mathbf{P}, \mathbf{M}) \in S \times \{\bar{\mathbf{M}}\}\}$$

But $A(\bar{\mathbf{P}}) = \{\bar{\mathbf{M}}\}$ because it is singleton, therefore

$$f(\bar{\mathbf{P}}, \bar{\mathbf{M}}) \leq \min\{f(\mathbf{P}, \bar{\mathbf{M}}) \mid (\mathbf{P}, \mathbf{M}) \in S \times A(\bar{\mathbf{P}})\}$$

Which is exactly the condition for the local optimality of theorem 4.2.8, thus, $(\bar{\mathbf{P}}, \bar{\mathbf{M}})$ is a local minimum for F. □

Given this last result, we can analyse under which circumstances $A(\bar{\mathbf{P}})$ is singleton. We will restrict our analyses to the special cases when $D(\cdot, \cdot)$ is the squared euclidean norm ($D(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2_{L^2}$) and when it is generally the L^p norm ($D(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_{L^p}$ - notice that $\| \cdot \|$ is not squared).

We will decompose f as a sum of k simpler functions, and define k sets $A_j(\bar{\mathbf{P}}_j)$, one for each column of $\mathbf{P}$. This we will find conditions regarding whether each $A_j(\bar{\mathbf{P}}_j)$ is singleton or not, which is related to the singleness of $A(\bar{\mathbf{P}})$. So we define for each $j = 1, 2, \cdots k$

$$f_j(\mathbf{p}_j, \mathbf{m}_j) = \sum_{i=1}^{n} (\mathbf{p}_j)_i D(\mathbf{x}_i, \mathbf{m}_j)$$

Hence

$$f(\mathbf{P}, \mathbf{M}) = \sum_{i=1}^{n} \sum_{j=1}^{k} p_{ji} \; D(\mathbf{x}_i, \mathbf{m}_j) = \sum_{i=1}^{k} f_j(\mathbf{p}_j, \mathbf{m}_j)$$

Where $\mathbf{p}_j = (p_{j1}, p_{j2}, \cdots p_{jn})$ and $\mathbf{m}_j$ are both the $j-th$ column of $\mathbf{P}$ and $\mathbf{M}$ respectively.

For each $j = 1, 2, \cdots k$, $\mathbf{m}_j$ lies in the set of all convex combinations of the $\mathbf{x}_i$'s (the convex hull of the data set D), and more specifically, given a value for $\mathbf{P} \in S$ in an extreme of S, $\mathbf{m}_j$ can be restricted to lie in the convex hull of $C_i = \{\mathbf{x}_l \mid p_{il} = 1\}$. So, we can assume that $\mathbf{p}_j \in V_j$, a compact set.

In these conditions we can define $A_j(\mathbf{p}_j)$:

Definition 4.2.5. $A_j(\mathbf{p}_j) = \{\mathbf{m}_j \in \mathbb{R}^{\mathrm{d}} \mid f_j(\mathbf{p}_j, \mathbf{m}_j) \leq f_j(\mathbf{p}_j, \mathbf{m}) \;\; \forall \; \mathbf{m} \in V_j\}$

Theorem 4.2.10. *Let $\bar{\mathbf{P}} = (\mathbf{p}_1, \mathbf{p}_2, \cdots {}_k)^T$. Then $A(\bar{\mathbf{P}})$ is singleton if and only if each $A_j(\mathbf{p}_j)$ is singleton*

Proof. For each j, pick a value $\hat{\mathbf{m}}_j \in A_j(\mathbf{p}_j)$. By definition, $f_j(\mathbf{p}_j, \mathbf{m}_j) \leq f_j(\mathbf{p}_j, \mathbf{m})$ for all $\mathbf{m} \in \mathbb{R}^{\mathrm{d}}$, so we must have

$$\begin{gathered} f(\bar{\mathbf{P}}, (\hat{\mathbf{m}}_1, \hat{\mathbf{m}}_2, \cdots \hat{\mathbf{m}}_k)) = \sum_{j=1}^{k} f_j(\mathbf{p}_j, \hat{\mathbf{m}}_j) \leq \sum_{j=1}^{k} f_j(\mathbf{p}_j, \mathbf{m}_j) = f(\bar{\mathbf{P}}, \mathbf{M}) \\ \text{for all } \mathbf{M} = (\mathbf{m}_1, \mathbf{m}_2, \cdots \mathbf{m}_k) \in \mathcal{M}_{n \times k}(\mathbb{R}) \end{gathered} \tag{4.6}$$

Hence $(\hat{\mathbf{m}}_1, \hat{\mathbf{m}}_2, \cdots \hat{\mathbf{m}}_k) \in A(\bar{\mathbf{P}})$.

If any $A_j(\mathbf{p}_j)$ has more than one element, pick two of them $\hat{\mathbf{m}}_j$ and $\hat{\mathbf{m}}'_j$. Thus, we could define

$$\begin{aligned} \mathbf{M} &= (\hat{\mathbf{m}}_1, \cdots \hat{\mathbf{m}}_j \cdots \hat{\mathbf{m}}_k) \\ \mathbf{M}' &= (\hat{\mathbf{m}}_1, \cdots \hat{\mathbf{m}}'_j \cdots \hat{\mathbf{m}}_k) \end{aligned}$$

Both satisfy condition 4.4 hence $A(\bar{\mathbf{P}})$ has at least these two different elements. This proves the assertion.

□

Finally we are prepared to analyse the partial optimal solutions for when $D(\cdot, \cdot)$ is the L^p norm and the squared euclidean norm, through the following two theorems:

Theorem 4.2.11. *If* $(\bar{\mathbf{P}}, \bar{\mathbf{M}})$ *is a partial optimal solution of problem 4.1 for the case when* $D(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2_{L^2}$*, then it is a local minimum point.*

Proof. Decomposing f in k functions we will prove that each $A_j(\bar{\mathbf{p}}_j)$ is singleton:

$$\begin{aligned} f(\bar{\mathbf{P}}, \bar{\mathbf{M}}) &= \sum_{j=1}^{k} f_j(\bar{\mathbf{p}}_j, \bar{\mathbf{m}}_j) \\ \text{where} \quad f_j(\mathbf{p}_j, \mathbf{m}_j) &= \sum_{i=1}^{n} p_{ji} \|\mathbf{x}_i - \mathbf{m}_j\|^2_{L^2} \end{aligned}$$

The norm L^2 is strictly convex (because the second derivative of the function $h(\mathbf{z}) = \|\mathbf{z}\|^2_{L^2}$ is strictly positive definite), hence for every $j = 1, 2, \cdots k$, $t \in (0, 1)$ and $\mathbf{m},\ \mathbf{n} \in \mathbb{R}^{\mathrm{d}}$ we have:

$$\begin{aligned} f_j(\mathbf{p}_j, t\mathbf{m} + (1-t)\mathbf{n}) &= \sum_{i=1}^{n} p_{ji} \|(t\mathbf{x}_i + (1-t)\mathbf{x}_i) - (t\mathbf{m} + (1-t)\mathbf{n})\|^2_{L^2} = \\ &= \sum_{i=1}^{n} p_{ji} \|t(\mathbf{x}_i - \mathbf{m}) + (1-t)(\mathbf{x}_i - \mathbf{n})\|^2_{L^2} < \\ &< \sum_{i=1}^{n} p_{ji} (t\|\mathbf{x}_i - \mathbf{m}\|^2_{L^2} + (1-t)\|\mathbf{x}_i - \mathbf{n}\|^2_{L^2} = \\ &= t(\sum_{i=1}^{n} p_{ji} \|\mathbf{x}_i - \mathbf{m}\|^2_{L^2}) + (1-t)(\sum_{i=1}^{n} p_{ji} \|\mathbf{x}_i - \mathbf{n}\|^2_{L^2}) = \\ &= t f_j(\mathbf{p}_j, \mathbf{m}) + (1-t) f_j(\mathbf{p}_j, \mathbf{n}) \end{aligned}$$

Which means that f_j is strictly convex on the second argument. Now, given a fixed value $\bar{\mathbf{p}}_j$, we show that there can be only one minimum, for suppose there exists two values $\mathbf{m}_1 \neq \mathbf{m}_2$ such that both satisfy

$$f_j(\bar{\mathbf{p}}_j, \mathbf{m}_l) \leq f_j(\bar{\mathbf{p}}_j, \mathbf{m}) \quad \text{for all } \mathbf{m} \in \mathbb{R}^{\mathrm{d}} \text{ and } l = 1, 2 \tag{4.7}$$

Clearly,

$$f_j(\bar{\mathbf{p}}_j, \mathbf{m}_1) \leq f_j(\bar{\mathbf{p}}_j, \mathbf{m}_2) \leq f_j(\bar{\mathbf{p}}_j, \mathbf{m}_1) \Rightarrow f_j(\bar{\mathbf{p}}_j, \mathbf{m}_2) = f_j(\bar{\mathbf{p}}_j, \mathbf{m}_2)$$

So that combining 4.7 and the strict convexity we conclude a contradiction

$$\begin{aligned} f_j(\bar{\mathbf{p}}_j, \mathbf{m}_1) \leq f_j(\bar{\mathbf{p}}_j, (1-t)\mathbf{m}_1 + t\mathbf{m}_2) < (1-t) f_j(\bar{\mathbf{p}}_j, \mathbf{m}_1) + t f_j(\bar{\mathbf{p}}_j, \mathbf{m}_2) = \\ = f_j(\bar{\mathbf{p}}_j, \mathbf{m}_1) + t(f_j(\bar{\mathbf{p}}_j, \mathbf{m}_2) - f_j(\bar{\mathbf{p}}_j, \mathbf{m}_1)) = f_j(\bar{\mathbf{p}}_j, \mathbf{m}_1) \end{aligned}$$

Thus, there can exist only one minimum for $\mathbf{m}$, given a fixed $\bar{\mathbf{p}}_j$.
Let's calculate the first and second derivatives for f_j at $(\bar{\mathbf{p}}_j, \bar{\mathbf{m}}_j)$:

$$f_j(\bar{\mathbf{p}}_j, \mathbf{m}) = \sum_{i=1}^{n} \bar{p}_{ji} \sum_{l=1}^{d} (x_{il} - m_l)^2$$

$$\Rightarrow \begin{cases} \frac{\partial f_j}{\partial m_l}(\bar{\mathbf{p}}_j, \mathbf{m}) = -2 \sum\limits_{i=1}^{n} \bar{p}_{ji}(x_{il} - m_l) \\ \frac{\partial^2 f_j}{\partial m_l \partial m_{l'}}(\bar{\mathbf{p}}_j, \mathbf{m}) = 2(\sum\limits_{i=1}^{n} \bar{p}_{ji}) \delta(l, l') \end{cases}$$

So, the hessian of f_j is strictly convex:

$$H(\bar{\mathbf{p}}_j, \mathbf{m}) = 2(\sum_{i=1}^{n} \bar{p}_{ji}) \mathbf{I} > \mathbf{0}$$

Which means that equating to zero the derivative gives the minimum of f_j, giving:

$$\begin{aligned} 0 = \frac{\partial f_j}{\partial m_l}(\bar{\mathbf{p}}_j, \mathbf{m}) &= \sum_{i=1}^{n} \bar{p}_{ji}(x_{il} - m_l) \\ \Rightarrow \mathbf{m}_{min} &= \frac{\sum_{i=1}^{n} \bar{p}_{ji} \mathbf{x}_i}{\sum_{i=1}^{n} \bar{p}_{ji}} \end{aligned}$$

But this is the unique minimum, hence $\bar{\mathbf{m}}_j = \mathbf{m}_{min} \in A_j(\mathbf{p}_j)$ and each $A_j(\bar{\mathbf{p}}_j)$ is singleton, so the assertion is proven, considering theorem 2.6.16.

□

Finally we do a similar analysis for when $D(\cdot, \cdot)$ is given by the L^p norm:

Theorem 4.2.12. *Consider the norm L^p a point $\bar{\mathbf{P}}$ made of 0's and 1's, and $A_j(\bar{\mathbf{p}}_j)$. Then $A_j(\bar{\mathbf{p}}_j)$ is non-singleton if and only if the following conditions hold simultaneouly:*

i. The points $\mathbf{x}_i$ are all collinear;

ii. $\sum\limits_{l=1}^{k} p_{lj}$ *is even.*

Proof. Assume both conditions (i) and (ii) holds. Then, by condition (i), there exists $\mathbf{a}$, $\mathbf{b}$, and t_i's such that $\mathbf{x}_i = \mathbf{a} + t_i\mathbf{b}$ for $i = 1, 2, \cdots n$. $A_j(\bar{\mathbf{p}}_j)$ is the set of values $\mathbf{m}$ that minimize f_j given the fixed value $\bar{\mathbf{p}}_j$, which has the expression

$$f_j(\bar{\mathbf{p}}_j, \mathbf{m}) = \sum_{i=1}^{n} p_{ij} \|\mathbf{a} + t_i\mathbf{b} - \mathbf{m}\|_{L^p} \tag{4.8}$$

$\mathbf{m}_{min}$ lies in the *convex hull* formed by the collinear vectors $\mathbf{a} + t_i\mathbf{b}$, so we should find a minimum through the points of the form $\mathbf{a} + s\mathbf{b}$. We can simplify the problem 4.8 to minimization of the function

$$\begin{aligned} h(s) = f_j(\bar{\mathbf{p}}_j, \mathbf{a} + s\mathbf{b}) &= \sum_{i=1}^{n} p_{ij}\|\mathbf{b}\|_{L^p}|t_i - s| \\ \Rightarrow h(s) \equiv \sum_{i=1}^{n} \alpha_i |s - t_i| = & \\ = \sum_{l=1}^{i-1} \alpha_l (s - t_l) + \sum_{l=i}^{n} \alpha_l (t_l - s) \ , & \quad \text{if } s \in (t_{i-1}, t_i) \end{aligned} \tag{4.9}$$

Where $\alpha_i := p_{ij}\|\mathbf{b}\|_{L^p}$.
By 4.9, if $s \in (t_{i-1}, t_i)$,

$$\frac{dh}{ds}(s) = \sum_{l=1}^{i-1} \alpha_l - \sum_{l=i}^{n} \alpha_l \equiv d_i$$

We extend this definition for when $s \in (-\infty, t_1)$ and $s \in (t_n, \infty)$ respectively:

$$d_1 = -\sum_{l=1}^{n} \alpha_l \text{ and } d_{n+1} = \sum_{l=1}^{n} \alpha_l$$

It is readily seen that $d_1 < 0$, $d_n > 0$ and $d_{i+1} = d_i + 2\alpha_i \geq d_i$, for each i, hence there exists a i_0 such that

$$d_{i_0} \leq 0 < d_{i_0+1} \ \text{ or } \ d_{i_0} < 0 \leq d_{i_0+1} \tag{4.10}$$

As each d_i is the slope of a linear function connecting $h(t_{i-1})$ and $h(t_i)$, which is a continuous function, hence t_{i_0} is a minimum of h.
Therefore, a solution of 4.8 is

$$\mathbf{m}_{min} = \mathbf{a} + t_{i_0}\mathbf{b}$$

Where i_0 some only index that satisfies 4.10, which is the same as to satisfy one of the following conditions

$$\begin{gathered}\sum_{l=1}^{i_0-1} p_{lj} < \sum_{l=i_0}^{n} p_{lj} \quad \text{and} \quad \sum_{l=1}^{i_0} p_{lj} \geq \sum_{i_0+1}^{n} p_{lj} \\ \text{or} \\ \sum_{l=1}^{i_0-1} p_{lj} \leq \sum_{l=i_0}^{n} p_{lj} \quad \text{and} \quad \sum_{l=1}^{i_0} p_{lj} < \sum_{i_0+1}^{n} p_{lj}\end{gathered} \tag{4.11}$$

(the minimum of this function on the line $\mathbf{a} + t\mathbf{b}$ is called the geometric median)
Here $p_{ij} \in \{0, 1\}$, and by condition (ii), $\sum_{i=1}^{n} p_{ij} = 2N$. Consider

$$m_0 = \min \left\{ m \;\middle|\; \sum_{i=1}^{m} p_{ij} = N + 1 \right\}$$

Then

$$\begin{aligned} d_{m_0} &= \|\mathbf{b}\|_{L^p} \left(\sum_{i=1}^{m_0-1} p_{ij} - \sum_{i=m_0}^{n} p_{ij} \right) = \|\mathbf{b}\|_{L^p}(N - (2N - N)) = 0 \\ d_{m_0+1} &= \|\mathbf{b}\|_{L^p} \left(\sum_{i=1}^{m_0} p_{ij} - \sum_{i=m_0+1}^{n} p_{ij} \right) = \|\mathbf{b}\|_{L^p}(N + 1 - (2N - (N + 1))) = 2\|\mathbf{b}\|_{L^p} \end{aligned}$$

Hence

$$d_{m_0} \leq 0 < d_{m_0+1}$$

Now consider

$$n_0 = \min \left\{ m \;\middle|\; \sum_{i=1}^{m} p_{ij} = N \right\}$$

Then

$$\begin{aligned} d_{n_0} &= \|\mathbf{b}\|_{L^p} \left(\sum_{i=1}^{n_0-1} p_{ij} - \sum_{i=n_0}^{n} p_{ij} \right) = \|\mathbf{b}\|_{L^p}(N - 1 - (2N - (N - 1))) = -2\|\mathbf{b}\|_{L^p} \\ d_{n_0+1} &= \|\mathbf{b}\|_{L^p} \left(\sum_{i=1}^{n_0} p_{ij} - \sum_{i=n_0+1}^{n} p_{ij} \right) = \|\mathbf{b}\|_{L^p}(N - (2N - N)) = 0 \end{aligned}$$

Which proves that $n_0 \neq m_0$ and

$$d_{n_0} < 0 \leq d_{n_0+1}$$

Therefore $\mathbf{a} + \mathbf{b}t_{m_0}$ and $\mathbf{a} + \mathbf{b}t_{n_0}$ are two minima of the function $f(\bar{\mathbf{p}}_j, \cdot)$ on V thus belonging to $A(\bar{\mathbf{p}}_j)$, making it non-singleton.

On the converse, assume $A(\bar{\mathbf{p}}_j)$ is non-singleton. If (i) is not valid and (ii) is true, then one can define n_0 as

$$n_0 = \max\{m \mid \sum_{i=1}^{m} p_{ij} = N + 1\}$$

and check that by the oddity of the above sum, this is the only index satisfying condition 4.11, thus making $A(\bar{\mathbf{p}}_j)$ singleton, a contradiction.
Whenever (i) is not valid then at least one $\mathbf{x}_i$ escapes the collinear condition, so that whatever two points $\mathbf{m}_1$ and $\mathbf{m}_2$ we select from the convex hull V formed by $\{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n\}$, there will always have at least one $\mathbf{x}_i$ not belonging to the line connecting these two selected points. Therefore the interior of V cannot be empty.
Suppose $\mathbf{m}_1$ and $\mathbf{m}_2$, with $\mathbf{m}_1 \neq \mathbf{m}_2$, lie within the interior of V so that these are not collinear with any of the $\mathbf{x}_i$'s. Now, the triangle inequality for $\|\cdot\|_{L^p}$ can be made strict, so we have

$$\begin{aligned} D(\mathbf{x}_i, (1-t)\mathbf{m}_1 + t\mathbf{m}_2) &= \|\mathbf{x}_i - (1-t)\mathbf{m}_1 - t\mathbf{m}_2\|_{L^p} \\ &= \|(1-t)(\mathbf{x}_i - \mathbf{m}_1) + t(\mathbf{x}_i - \mathbf{m}_2)\|_{L^p} < \\ &< |1-t|\|\mathbf{x}_i - \mathbf{m}_1\|_{L^p} + |t|\|\mathbf{x}_i - \mathbf{m}_2\|_{L^p} = \\ &= (1-t)D(\mathbf{x}_i, \mathbf{m}_1) + tD(\mathbf{x}_i, \mathbf{m}_2) \end{aligned}$$

Hence for every i such that $p_{ij} = 1$, $D(\mathbf{x}_i, \cdot)$ is strictly concave, making f_i strictly concave in the interior of V. Now, if there is a minimum for f_i, it cannot lie in the border ∂V (because of the strict concavity condition proven above), so it belongs necessarily to the interior of V, and by the strict inequality, it should be unique, contradicting the hypothesis that $A_i(\bar{\mathbf{p}}_i)$ is non-singleton.

□

Chapter 5

Clustering in high dimensions

Until now we analyzed in detail the K-Means algorithm, which will be the core of the final application of this thesis. But this algorithm relies heavily on the measurement of the distance between a point and its closest updated cluster center, *i.e.*, on the quantities $\|\mathbf{x} - \mathbf{m}_i\|_{L^p}$ and more generally $D(\mathbf{x}, \mathbf{m}_i)$ for any specific *dissimilarity measure.* Many other cluster techniques like agglomerative clustering, density based methods, many hierarchical models, graph clustering rely on the calculation of distances between data points. In fact many Supervised Learning methods rely heavily as well on the calculation of distance between pairs of points, like the very popular *k-nearest neighbours* algorithm.
The choice of the good norms or dissimilarities is surely an important problem that will not be treated here, but once one such distance notion is chosen, quite often we have to handle with a troublesome phenomenon in computing: the *concentration of measure.* We illustrate it with some known simple examples and then discuss why it can endanger the effectiveness of our algorithms.
We start with a classical example of concentration of measure, the

Lemma 5.0.1 (Hoeffding's inequality)**.** *Let $X_1, X_2, \cdots X_n$ be independent variables such that $a_i \leq X_i \leq b_i$. Then*

$$P\left(\left| \frac{X_1 + X_2 + \cdots + X_n}{n} - \mathbb{E}\left[\frac{X_1 + X_2 + \cdots + X_n}{n}\right] \right| \geq \epsilon \right) \leq 2\mathrm{e}^{-2\epsilon^2 n^2 / \sum\limits_{i=1}^{n} (b_i - a_i)^2}$$

Basically it says that as $n \to \infty$, $\frac{X_1+X_2+\cdots+X_n}{n} \to \mathbb{E}[\frac{X_1+X_2+\cdots+X_n}{n}]$ in probability.
One can think of an n-dimensional random variable $Y = (X_1, X_2, \cdots, X_n)$ and the function $\varphi(Y) = \frac{X_1+X_2+\cdots+X_n}{n}$ so that the result can be recast as

$$P(|\varphi(Y) - \mathbb{E}[\varphi(Y)]| \geq \epsilon) \overset{n\to\infty}{\longrightarrow} 0 \quad \forall\, \epsilon > 0$$

This result can be extended for general 1-Lipschitz functions in *product measure spaces* see reference [11].
To picture how the consequences of this lemma could naturally arise in a computational problem, suppose there are data points which are the outcomes of a random variable $X \in [-1, 1]^d$

uniformly distributed over each interval $[-1, 1]$. Define the new variable $Y = (X_1^2, X_2^2, \cdots X_d^2)$ and the function φ as before, and calculate

$$\mathbb{E}[\varphi(Y)] = \frac{1}{d}\sum_{i=1}^{d}\mathbb{E}\left[X_i^2\right] = \frac{1}{d}\sum_{i=1}^{d}\int_{-1}^{1}\frac{x^2}{2}dx = \frac{1}{d}\left(\frac{d}{3}\right)$$

Noticing that $\varphi(Y) = \|X\|_{L^2}^2/d$, we apply the *Hoeffding's inequality* lemma:

$$P(|\varphi(Y) - \mathbb{E}[\varphi(Y)]| \geq \epsilon) = P(|\|X\|_{L^2}^2 - d/3| \geq \epsilon d) \leq 2\mathrm{e}^{-2\epsilon^2 d^2/\sum\limits_{i=1}^{d}(1-0)^2} = 2\mathrm{e}^{-2\epsilon^2 d}$$

Now, we will work with $d = 100$ and $\epsilon = 0.1$. Then

$$P(|\|X\|_{L^2}^2 - 100/3| \geq 10) \leq 2\mathrm{e}^{-2} \approx 0.27$$

Hence

$$P(|\|X\|_{L^2}^2 - 100/3| < 10) = P(100/3 - 10 < \|X\|_{L^2}^2 < 100/3 + 10) \approx$$

$$\approx P(23 < \|X\|_{L^2}^2 < 43) = P(\sqrt{23} < \|X\|_{L^2} < \sqrt{43}) \approx 0.73$$

This means that in approximately 73% of the data lies in the interval $(4.8, 6.6)$, if compared to $\sqrt{\mathbb{E}[\|X\|_{L^2}^2]} = \sqrt{100/3} \approx 5.8$, this fact can dramatically affect distance measurements between pairs of points - most of the points will be approximately equidistant so that the clustering algorithms will not be effective to separate them in special clearly separated regions.
In Data Science a typical data set can have tenths of features, and quite often we have to tackle data points which are represented as vectors with dimensions up to $d = 100$ or even $d = 1000$. Let's imagine this terrible scenario of $d = 1000$ and $\epsilon = 0.1$ in which case a small calculation would give

$$P(|\|X\|_{L^2}^2 - 1000/3| \geq 1) \leq 2\mathrm{e}^{-20} \approx 4 \times 10^{-9}$$

Therefore

$$P(\sqrt{332} < \|X\|_{L^2} < \sqrt{334}) \approx 1 - 4 \times 10^{-9}$$

Now we ilustrate *concentration of measure* phenomenon with an experiment: we generate random vectors of crescent dimensions - here we will see $d = 2, 7, 30, 60$ and 120 - then we plot the histogram of the L^2 distances between all the pairs of points in a sample of 1000 points, shown in Figure 5.1. These points were generated using a gaussian distribution with covariance values selected randomly between 0 and 10, and mean $(15, 15, \cdots)$ (these parameters were chosen arbitrarily). After the calculation of theses distances, they were normalized to have mean zero and variance one (by subtracting the actual mean and then dividing by the actual variance) - this is necessary in order to compare the different cases in the same plot. Also the histograms were normalized to represent actual probability density functions.

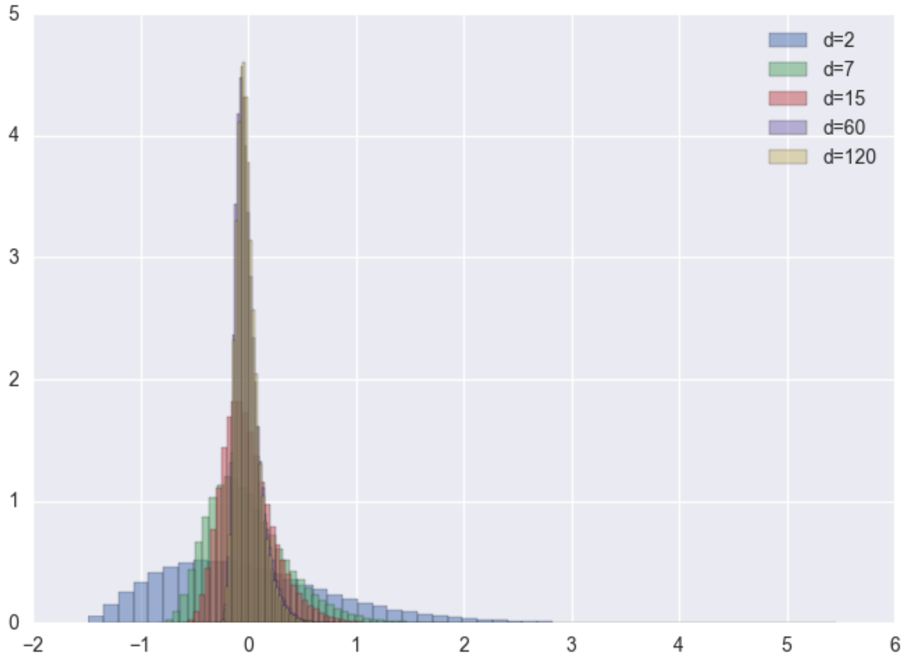


Figure 5.1: Histogram for the distance between all pairs of points in a sample of 1000 randomly generated vectors n dimensions $d = 2, 7, 15, 60, 120$. Notice that for $d = 60$ and 120 the histograms have almost the same shape.

In this image it can be seen that the histogram has a broaden aspect when the dimension d is small and gets sharper as dimension d grows, also we notice that there is practically no difference between the cases $d = 60$ and $d = 120$. This means that when d is small it can be distinguished a broad range of values for the distance between pairs of points, while when it grows, the range o possible values gets narrower so that in probability the distances are approximately all equal.

Let's put this observation on solid grounds by proving some interesting results about concentration of measures in the context of measuring distances using L^p norms. Next we state and prove some ilustrative results *concentration of measure* and give some empirical evidence for it through images $5.1, 5.2, 5.3$.

5.1 An example of concentration of measure

We present some results that ilustrate the phenomenon of *concentration of measure* due to Gérard Biau and David M. Mason in [7].We start with the

Proposition 5.1.1. *Let $\{Y_d\}_{d\geq 1}$ be a sequence of random variables such that $Y_d \xrightarrow{P} a$ (Y_d converges in probability to a constant a), and let φ be a real-valued measurable function which* *is continuous at a. Assume that the following conditions hold:*

i. *φ is bounded on $[-M, M]$ for some $M > |a|$;*

ii. *$E[\ |\varphi(Y_d)|\ ] < \infty$ for all $d \geq 1$.*

Then,

$$\lim_{d\to\infty} E[\varphi(Y_d)] = \varphi(a) \quad \textit{if, and only if,} \quad \lim_{d\to\infty} E[\varphi(Y_d) I_{[\ |Y_d|>M\ ]}] = 0 \tag{5.1}$$

.

(I_A is the indicator function for the set A, defined by $I_A(x) = \begin{cases} 1 & \textit{if} \quad x \in A \\ 0 & \textit{if} \quad x \notin A \end{cases}$).

Proof. Define the random variables $X_d = \varphi(Y_d) I_{[\ |Y_d| \leq M\ ]}$. By continuity of φ at a and condition $i.$, $X_d \xrightarrow{P} \varphi(a)$ and $|X_d| \leq K$ for some k that bounds φ on $[-M, M]$, so the variables X_d fulfill the conditions for the *dominated convergence theorem* (see [31]), which gives us

$$\lim_{d\to\infty} E[X_d] = \lim_{d\to\infty} E[\ \varphi(Y_d) I_{[\ |Y_d| \leq M\ ]}\ ] = \varphi(a)$$

Now the result follow easily from equality

$$E[\varphi(Y_d)] = E\left[\ \varphi(Y_d) I_{[\ |Y_d|>M\ ]}\ \right] + E\left[\ \varphi(Y_d)\ I_{[\ |Y_d| \leq M\ ]}\ \right]$$

□

Our interest is to investigate what happens with the expectation and variance of

$$\|X\|_{L^p} = \sqrt[p]{\sum_{i=1}^{n} |X_i|^p} = \sqrt[p]{n} \sqrt[p]{\sum_{i=1}^{n} |X_i|^p} =: \sqrt[p]{n}\ \varphi\left(\sum_{i=1}^{n} |X_i|^p\right)$$

Therefore, we are interested in applying the previous result for the random variables

$$Y_d = \frac{1}{n} \sum_{i=d}^{d} W_i$$

Where $W_1, W_2, \cdots W_d$ are d independent identically distributed (*iid*) random variables with finite mean $E[W_i] = \mu$ and d is the dimension of the state space where data points $W = (W_1, W_2, \cdots W_d)$ are taken from. By the *Law of Large Numbers* (see [31]), as $E[W_i] = \mu$, then $Y_d \xrightarrow{P} \mu$. Now let's check when condition 5.1 holds:

Lemma 5.1.1. *Let φ be real-valued measurable function, $W_1, W_2, \cdots W_d$ iid with the same distribution of a random variable W (say, W_1), and assume one of the following conditions hold:*

i. *the function $|\varphi|$ is convex on $\mathbb{R}$ and $E[\ |\varphi(W)|\ ] < \infty$.*

ii. *For some $s > 1$, $\limsup_{d\to\infty} E[\ |\varphi(Y_d)|^s\ ] < \infty$.*

Then 5.1 is satisfied for $\{Y_d = \frac{1}{n}\sum_{i=d}^{d} W_i\}_{d=1,2,\cdots}$ *with* $a = \mu$ *and* $M > |\mu|$.

Proof. If *i.* is satisfied, by the convexity assumption, $|\varphi(\frac{1}{n}\sum_{i=1}^{d} W_i)| \leq \sum_{i=1}^{d} \frac{1}{n}|\varphi(W_i)|$, therefore

$$E[\ |\varphi(Y_d)|I_{[\ |Y_d|>M\ ]}\ ] \leq \frac{1}{d}E[\ |\varphi(W_i)|I_{[\ |Y_d|>M\ ]}\ ] = E[\ |\varphi(W)|I_{[\ |Y_d|>M\ ]}\ ]$$

But $Y_d \xrightarrow{P} \mu$ and $|M| > \mu$, $I_{[\ |Y_d|>M\ ]} \xrightarrow{P} 0$ so it holds the hypothesis for the dominated convergence theorem:

$$|\varphi(W)|I_{[\ |Y_d|>M\ ]} \xrightarrow{P} 0 \quad \text{and} \quad |\varphi(W)|I_{[\ |Y_d|>M\ ]} \leq |\varphi(W)|$$

Hence, $E\ |\varphi(W)|I_{[\ |Y_d|>M\ ]}\ ] \to 0$.
Assume now that *ii.* is satisfied. We use Holder's inequality for $1/r + 1/s = 1$ with variables $\varphi(Y_d)$ and $I_{[\ |Y_d|>M\ ]}$. Noticing that $|I_{[\ |Y_d|>M\ ]}|^r = I_{[\ |Y_d|>M\ ]}$ we arrive at the inequality

$$\begin{aligned} E[\ |\varphi(Y_d)|I_{[\ |Y_d|>M\ ]}\ ] &\leq (E[\ |\varphi(Y_d)^s|\ ])^{1/s}(E[\ I_{[\ |Y_d|>M\ ]}\ ])^{1/r} = \\ &= (E[\ |\varphi(Y_d)^s|\ ])^{1/s}P([\ |Y_d| > M\ ])^{1/r} \end{aligned}$$

But $P([\ |Y_d| > M\ ]) \xrightarrow{d\to\infty} 0$, which makes condition 5.1 true when combined with the above inequality plus the assumption *ii.*.

□

Corollary 5.1.1.1. *For fixed* $p > 0$ *and* $r > 0$, *consider for each* d, *the random vector* $\vec{X} = (X_1, X_2, \cdots X_d)$ *where the* X_i*'s are iid distributed as a random variable* X *(say,* X_1*).*

i. If $r/p < 1$ *and* $E[\ |X|^p\ ] < \infty$, *then*

$$\frac{E[\ \|\vec{X}\|_{L^p}^r\ ]}{d^{r/p}} \xrightarrow{d\to\infty} E[\ |X|^p\ ]^{r/p}$$

ii. If $r/p \geq 1$ *and* $E[\ |X|^r\ ] < \infty$, *then*

$$\frac{E[\ \|\vec{X}\|_{L^p}^r\ ]}{d^{r/p}} \xrightarrow{d\to\infty} E[\ |X|^p\ ]^{r/p}$$

Proof. We shall apply proposition 3.01 and lemma 3.02 to $W_i = |X_i|^p$, which are distributed as $W = |X|^p$, and $\varphi(u) = |u|^{r/p}$.
For *i.*, with $s = p/r > 1$ and $Y_d = \frac{1}{d}\sum_{i=1}^{d} |X_i|^p)$,

$$E[\ |\varphi(Y_d)|^s\ ] = E[\ (|Y_d|^{r/p})^{p/r}\ ] = E\left[\ \frac{1}{d}\sum_{i=1}^{d} |X_i|^p)\ \right] = E[\ |X|^p\ ] < \infty$$

Therefore, condition *ii.* of lemma 3.0.2 is satisfied giving in turn the result of proposition 3.0.1:

$$\frac{E[\ \|\vec{X}\|_{L^p}^r\ ]}{d^{r/p}} = E[\ (\frac{1}{d}\sum_{i=1}^{d} |X_i|^p)^{r/p}\ ] = E[\ \varphi(Y_d)\ ] \xrightarrow{d\to\infty} \varphi(E[\ |X|^p\ ]) = E[\ |X|^p\ ]^{r/p}$$

For *ii.*, defining $\varphi(u) = |u|^{r/p}$, because $r/p \geq 1$, so it is a convex function. Let's calculate

$$\begin{aligned} E[\ |\varphi(Y_d)|\ ] = E\left[\ \varphi\left(\sum_{i=1}^{d} \frac{1}{d}|X_i|^p\right)\ \right] \leq E\left[\ \sum_{i=1}^{d} \frac{1}{d}\varphi(|X_i|^p)\ \right] = \\ = \frac{1}{d}\sum_{i=1}^{d} E[\ |X_i|^r\ ] = E[\ |X|^r\ ] < \infty \end{aligned} \tag{5.2}$$

and also

$$E[\ |\varphi(|X|^p)|\ ] = E[\ |X|^r\ ] < \infty \tag{5.3}$$

The inequality 5.2 shows that the conditions of proposition 3.0.1 are fully satisfied and 5.3 shows that the condition *i.* of lemma 3.0.2 is satisfied, therefore, we conclude that

$$\frac{E[\ \|\vec{X}\|_{L^p}^r\ ]}{d^{r/p}} = E\left[\ \left(\frac{1}{d}\sum_{i=1}^{d}|X_i|^p\right)^{r/p}\ \right] = E[\ \varphi(Y_d)\ ] \overset{d\to\infty}{\longrightarrow} \varphi(E[\ |X|^p\ ]) = E[\ |X|^p\ ]^{r/p}$$

□

Now, let's apply this last result for $p > 0$ and $r = 1, 2$, and let's take for granted that $0 < E[\ |X|^m\ ] < \infty$ for $m = p, 1, 2$. So we conclude:

$$\frac{E[\ \|\vec{X}\|_{L^p}\ ]}{d^{1/p}} \overset{d\to\infty}{\longrightarrow} E[\ |X|^p\ ]^{1/p} \equiv \alpha$$

and

$$\frac{E[\ \|\vec{X}\|_{L^p}^2\ ]}{d^{2/p}} \overset{d\to\infty}{\longrightarrow} E[\ |X|^p\ ]^{2/p} \equiv \alpha^2$$

$$\therefore \quad \frac{\sqrt{Var[\ \|\vec{X}\|_{L^p}\ ]}}{E[\ \|\vec{X}\|_{L^p}\ ]} = \frac{\sqrt{E[\ \|\vec{X}\|_{L^p}^2\ ] - E^2[\ \|\vec{X}\|_{L^p}\ ]}}{E[\ \|\vec{X}\|_{L^p}\ ]} =$$

$$= \frac{\sqrt{d^{2/p}\{(E[\ \|\vec{X}\|_{L^p}^2\ ]/d^{2/p}) - (E[\ \|\vec{X}\|_{L^p}\ ]/d^{1/p})^2\}}}{d^{1/p}(E[\ \|\vec{X}\|_{L^p}\ ]/d^{1/p})} =$$

$$= \frac{\sqrt{(E[\ \|\vec{X}\|_{L^p}^2\ ]/d^{2/p}) - (E[\ \|\vec{X}\|_{L^p}\ ]/d^{1/p})^2}}{(E[\ \|\vec{X}\|_{L^p}\ ]/d^{1/p})} \overset{d\to\infty}{\longrightarrow}$$

$$\overset{d\to\infty}{\longrightarrow} \quad \frac{\sqrt{\alpha^2 - \alpha^2}}{\alpha} = 0$$

Thus we have concluded that $\frac{\sqrt{Var[\ \|\vec{X}\|_{L^p}\ ]}}{E[\ \|\vec{X}\|_{L^p}\ ]} \overset{d\to\infty}{\longrightarrow} 0$, which combined with *Chebyshev's inequality* gives us finally

$$P\left(\left|\frac{\|\vec{X}\|_{L^p}}{E[\ \|\vec{X}\|_{L^p}\ ]} - 1\right| \geq \epsilon\right) = P\left(\left|\|\vec{X}\|_{L^p} - E[\ \|\vec{X}\|_{L^p}\ ]\right| \geq \epsilon E[\ \|\vec{X}\|_{L^p}\ ]\right) \leq$$

$$\leq \frac{Var[\ \|\vec{X}\|_{L^p}\ ]}{\epsilon^2 E^2[\ \|\vec{X}\|_{L^p}\ ]} \overset{d\to\infty}{\longrightarrow} 0$$

This means that for small ϵ and sufficiently big d, the quantity $\|\vec{X}\|_{L^p}/E[\ \|\vec{X}\|_{L^p}\ ]$ fluctuates above ϵ with very small probability, almost zero, *i.e.*, the vector's lengths $\|\vec{X}\|_{L^p}$ measure more or less the same.

This result was proven for a special case where each component of the vector $\vec{X}$ is an *iid* variable. Below we check another experiment with vectors generated with the uniform distribution over the range from 0 to 1000 for each X_i, and then a thousand of these were selected for calculation of the relative distances using the L^2 norm. Below we display the histogram of these pairwise distances for dimensions $d = 2, 7, 15, 30, 60, 120$.

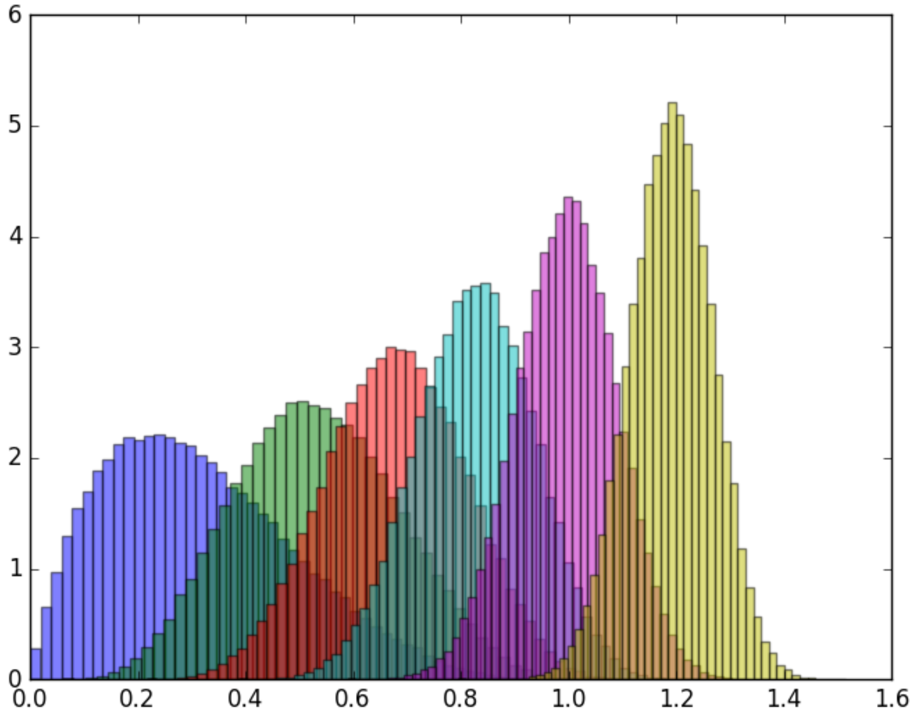


Figure 5.2: Superposed histograms showing the distribution of L^4 distance between random pairs vectors composed of $d = 2, 7, 15, 30, 60, 120$ *iid* components generated by a beta distribution

5.2 The Curse of Dimensionality

Clustering in high dimensions is problematic when the dimension d becomes big, because the quantity $\|\vec{X}\|_{L^p}$ becomes meaningless, *i.e.*, it cannot account well for devising which points are closer to a given cluster center, as more or less, every vector lies in a sphere of radius [$|\vec{X}\|_{L^p}$]. This phenomenon is intrinsic of the space and is reported in many computational situations, and it is blamed to be a *"Curse of Dimensionality"*.
This curse is not always intractable, but a warning should be done: there are some misconceptions of the consequences of this phenomenon as well as how different norms behave - for example, here the L^2 and L^1 norms are more robust in capturing distances as can be seen in the histogram pictured in Figure 5.3, for these cases the above result is still true though for d enormously large. Also, the results stated in the previous subsection are by no means the broader scenario, as they assume that the random vectors components are *iid*, which general is not true - in fact we expect that in general there are correlations between the different X_i's that we hope to capture through a redefinition of the features by an axis change through the so-called *Principal Component Analysis.*
For a good survey on these issues, refer to Zimek in [39].

5.3 Can we defeat it?

We cannot defeat it completely but some techniques are worth trying: reduction of dimension by disregarding non-informative components (like many *iid* X_i's which just add noise to the measurements); defining new variables from highly correlated X_i's which capture better the information in a smaller dimension; use of more robust distances if available; use of more external information and computing power to infer more complex relations among data points.
In this thesis two approaches were tried to takle these high dimensional issues: data dimension reduction through *Principal Component Analysis* (PCA) and implementation of a clustering algorithm called ORCLUS (by Charu Aggarwal) [2]. We describe these two methods in what follows along with results.

5.4 Principal Component Analysis

Consider a random variable $X\colon \Omega \to \mathbb{R}^d$ and its d components $X = (X_1, X_2, \cdots X_d)$. Given two such variables, X_i and X_j, their joint covariance is

$$cov(X_i, X_j) = E[\ (X_i - E[X_i])(X_j - E[X_j])\ ] = E[X_i X_j] - E[X_i]E[X_j]$$

Working with $Z = X - E[X]$,

$$Cov[X] = E[(X - E[X])^T (X - E[X])] = E[\ Z^T Z\ ]$$

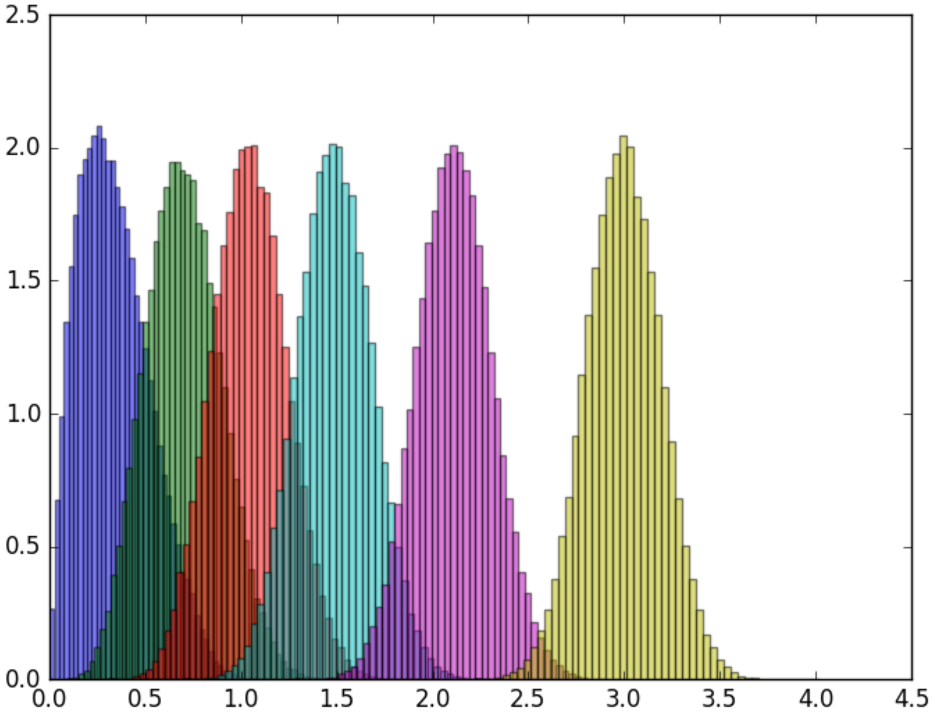


Figure 5.3: Superposed histograms showing the distribution of L^2 distance between random pairs vectors composed of $d = 2, 7, 15, 30, 60, 120$ *iid* components generated by a beta distribution. Notice that these histograms are more robust than those for L^4.

Suppose we change coordinates through a linear transformation $Z \mapsto PZ$. We check

$$\begin{aligned} E[\ (PZ)^T PZ\ ] &= \left[\ E[\ (\sum_{l=1}^{d} P_{il} Z_l)(\sum_{l'}^{d} P_{jl'} Z_{l'})\ ]\ \right]_{i,j=1,2,\cdots d} = \\ &= \left[\ \sum_{l=1}^{d} \sum_{l'=1}^{d} P_{il} E[\ Z_l Z_{l'})\ ] P_{jl'}\ \right]_{i,j=1,2,\cdots d} = \\ &= PE[\ Z^T Z\ ] P^T = P\ Cov[X]\ P^T \\ \therefore \quad Cov[X] &= P^T E[\ (PZ)^T PZ\ ] P \end{aligned} \tag{5.4}$$

We could think of a new set of variables $Y = PZ$ and a transformation P which diagonalizes the covariance matrix, so that in combination with 5.4,

$$\begin{aligned} & Cov[X] = P^T \; diag(\lambda_1, \lambda_2, \cdots \lambda_d) \; P = P^T \; E[\; Y^T Y \;] \; P \\ \therefore \quad & Cov[Y] = E[\; Y^T Y \;] = diag(\lambda_1, \lambda_2, \cdots \lambda_d) \end{aligned} \tag{5.5}$$

The new set of variables $Y_i = \sum_j P_{ij} Z_j$ have null joint covariance - in rough terms, for every realization $\vec{y} = (y_1, y_2 \cdots y_d)$ of Y, if $y_i y_j > 0$ (< 0) this will be balanced at some point by others realizations so to cancel it on average.
Each eigenvector $\vec{u}_i$ of $Cov[X]$ is a P row, *i.e.*, $\vec{u}_i = P^T \vec{e}_i$ and it is assumed that the λ's are decreasing. It is important to interpret them: according to 5.6, $\lambda_i = Var[Y_i]$, so that the bigger the i-th eigenvalue is, the more X fluctuates around the mean in the direction $\vec{u}_i$. Therefore one could look for this fluctuation around the most relevant eigenvalues instead of considering the whole data variatoin trhough all of its d dimensions. This motivates the construction of a projector-like operator $P^{(m)}$ next.
Select the $m < d$ bigger eigenvalues and define the projector operator

$$P^{(m)}(\vec{v}) = \sum_{l=1}^{m} \langle \vec{v}, P^T \vec{e}_l \rangle \; \vec{e}_l$$

and the reduced diagonal matrix

$$D^{(m)} = diag(\lambda_1, \lambda_2, \cdots \lambda_m)$$

The matrix $P^{(m)}$ is not exactly a projection operator, what it does instead is the following: Let $\vec{v}$ be expanded in the eigenvectors basis $\{P^T \vec{e}_1, P^T \vec{e}_2, \cdots P^T \vec{e}_d\}$, then $P^{(m)}$ sends $\vec{v}$ to the m first coefficients vector (a projector would instead send $\vec{v}$ to its truncation). In matrix terms, $P^{(m)}$ is composed of the first m lines of P.
We assert that the matrix $P^{(m)T} D^{(m)} P^{(m)}$ is the best approximation for $Cov[X]$ in the *Frobenius norm.* For instance,

$$P^{(m)T} D^{(m)} P^{(m)} = \left[\begin{array}{c|c|c|c} & & & \\ \vec{u}_1 & \vec{u}_2 & \cdots & \vec{u}_m \\ & & & \end{array} \right] \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_m \end{bmatrix} \left[\begin{array}{c} \vec{u}_1 \\ \hline \vec{u}_2 \\ \hline \vdots \\ \hline \vec{u}_m \end{array} \right] =$$

$$= \left[\begin{array}{c|c|c|c|c|c} \\ \vec{u}_1 & \vec{u}_2 & \cdots & \vec{u}_m & \cdots & \vec{u}_d \\ \\ \end{array}\right] \begin{bmatrix} \lambda_1 &&&&&& \\ & \lambda_2 &&&&& \\ && \ddots &&&& \\ &&& \lambda_m &&& \\ &&&& 0 && \\ &&&&& \ddots & \\ &&&&&& 0 \end{bmatrix} \left[\begin{array}{c} \vec{u}_1 \\ \hline \vec{u}_2 \\ \hline \vdots \\ \hline \vec{u}_m \\ \hline \vdots \\ \hline \vec{u}_d \end{array}\right] =$$

$$= \quad P^T diag(\lambda_1, \lambda_2, \cdots \lambda_m, 0, \cdots 0)P$$

$$\therefore \qquad \parallel P^T DP - P^{(m)T} D^{(m)} P^{(m)} \parallel_F = \parallel P^T diag(0, \cdots 0, \lambda_{m+1} \cdots \lambda_d)P \parallel_F = \sqrt{\lambda_{m+1}^2 + \cdots + \lambda_d^2}$$

(The Frobenius norm is defined as $\|A\|_F = \sqrt{\sum_i \sum_j |a_{ij}|^2} = \sqrt{tr(A^T A)} = \sqrt{\sum_i \sigma_i(A)^2}$ where the $\sigma_i(A)$'s are the eigenvalues of A)
Hence

$$\begin{cases} \|Cov[X] - P^{(m)T} D^{(m)} P^{(m)}\| = \sqrt{\lambda_{m+1}^2 + \cdots + \lambda_d^2} \\ \\ P^{(m)T} D^{(m)} P^{(m)} = P^T diag(\lambda_1, \lambda_2, \cdots \lambda_m, 0, \cdots 0)P \\ \\ \min \lambda = \lambda_d \leq \lambda_{d-1} \cdots \leq \lambda_{m+1} \end{cases} \tag{5.6}$$

5.6 tells that the matrix $P^{(m)}$ maps d-dimensional vectors into $\mathbb{R}^m$ minimizing the loss of information of $P^{(m)T} D^{(m)} P^{(m)}$ in relation to $Cov[X]$, in the sense that the first m most important eigenvectors are kept, while the remaining $d - m$ smallest eigenvalues are reset to zero. Now, how is it applied in the actual data set? We use the matrix $P^{(m)}$ to project each realization $\vec{x}$ into a smaller m-dimensional vector $P^{(m)}\vec{x}$. But $Cov[X]$ is replaced to the resized empirical covariance matrix. Assume the data set has mean zero (by subtracting the actual mean if necessary), and group realizations in a $n \times d$ matrix, with $n = \#D$:

$$\mathbf{X} = \left[\begin{array}{c} \vec{x}_1 \\ \hline \vec{x}_2 \\ \hline \vdots \\ \hline \vec{x}_n \end{array}\right]$$

The empirical covariance is proportional to $\mathbf{X}^T\mathbf{X}$, so we perform an Singular Value Decomposition (SVD) that should give:

$$\mathbf{X}^T\mathbf{X} = P^T DP$$

And we define the 'empirical' $P^{(m)}$ as the truncation of P up to m rows. Another possibility is to perform an SVD on $\mathbf{X}$, in which case the composition becomes $U^T \Sigma P$, where P is the same as above (but this is not a good option if n is huge).

Once $P^{(m)}$ is learnt, one can apply it on the full d-dimensional realizations to reduce their components, a process of which is called *reduction of dimensionality* - we keep the relevant dimensions of a properly rotated axis (eigenvectors) and discard those which only add noise (*i.e.*, have low variance). The calculation of *SVD* in order to truncate the matrix P which diagonalizes $\mathbf{X}^T\mathbf{X}$, in turn, reducing the dimensionality associated to the data is called *Principal Component Analysis* (*PCA*).

5.5 PCA + K-Means

We provided evidence that K-Means and any algorithm which relies on distance metrics can be highly affected by the *curse of dimensionality.* We can use PCA to reduce dimensionality and apply these algorithms for the new data in the relevant dimensions, *i.e.*, the realizations components around most relevant eigenvectors. As a matter of fact it is a standard procedure to preprocess the data set through a *Principal Component Analysis*, which almost always leads to better results in Machine Learning methods.
This technique is widely employed in Data Science, namely for *information retrieval systems*, where unstructured data sets like text files are transformed into vectors to be 'rotated' through *PCA* giving rise to new vector representations whose most important components are remarkably interpretated as the main covered topics. Also in Genetic Biology PCA is largely employed in order to point out the relevant correlations among the hundreds or thousands gene features to a DNA sequence, see [36]. For good references on advanced PCA techniques see the classical book [17]. The K-Means and the PCA are closely interconnected techniques as reported in [13] hence their combined use leads to excelent results in clustering data. But as will be seem at Section 5.7, this technique is not robust enough to fight the *curse of dimentionality* nor the *overfitting* of clusters around small sets of noisy points so-called *outliers.*

5.6 ORCLUS

The *dimension reduction* provided by PCA is generally very effective. But consider the following two examples of data sets in Figure 5.4. It displays two very characteristic clusters, one whose points are very correlated in the $x{\times}y$ plane, another in the z direction (both planes $x{\times}z$ and $y \times z$ fit well). A SVD was applied and the directions are shown, the least relevant being that pointing downwards in z. If we cut it, we will see dimension reduced points very similar to the projection $x \times y$ displayed at the bottom left - basically, the right cluster will collapse to a region with a tiny radius, losing a lot of relevant information (the different heights along z). A similar situation will happen for points in Figure 5.6. The clusters have very special shapes, and are fully described at different hiperplanes - the left in $x \times z$ and the right in $x \times y$. Even though PCA could reduce dimension and then K-Means could find these clusters, once they are relatively simple, on more complex data sets this loss of information described for Figure 5.4 can bring some difficulty in finding the natural clusters, for instance, consider the clusters in Figure 5.6. There, if the less relevant direction points downwards and it is cut, we will not be able to identify the two similar clusters on the right.

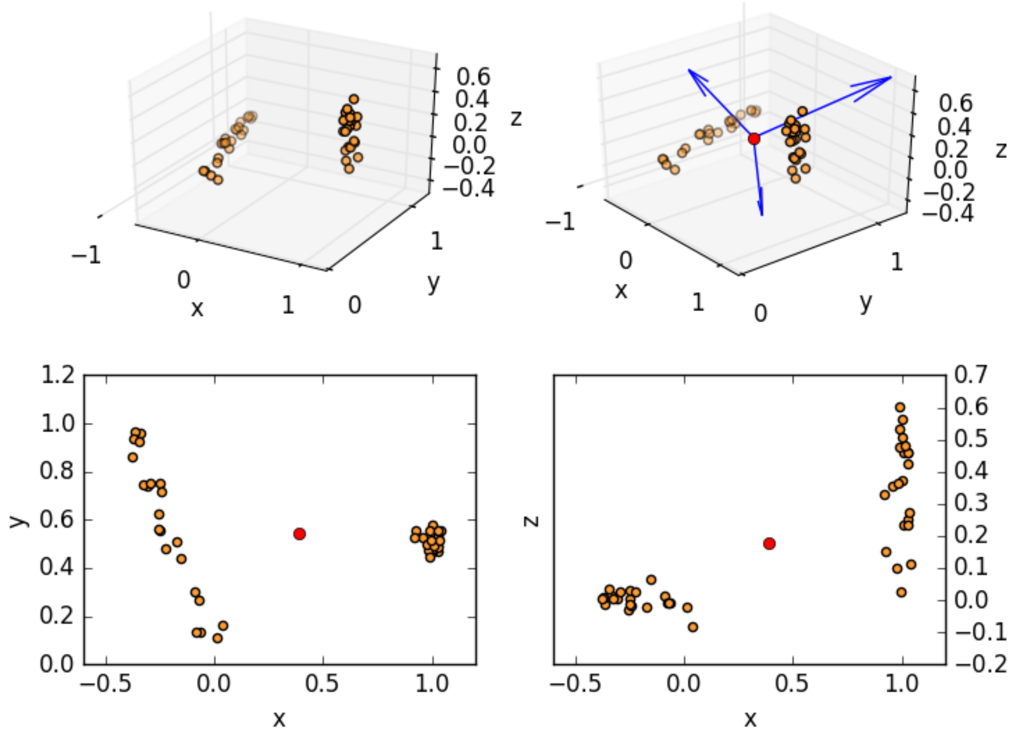


Figure 5.4: Two cross sections of the same set of points. The $x \times y$ cross section shows a left set collapsed to very small circular region, while the $x \times z$ shows it spread along the z direction. On the top right the SVD eigenvectors are displayed. The vector pointing downwards in $zdirection$ is the least relevant direction. Cutting it means to lose the natural structure of the cluster on the right.

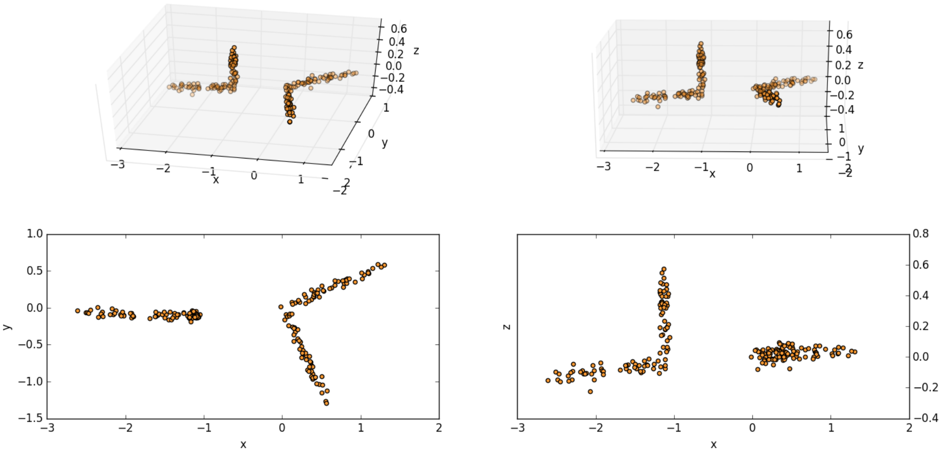


Figure 5.5: Two clusters living close to two different subdimensions. A full dimensional clustering procedure would incur in considerable loss of information.

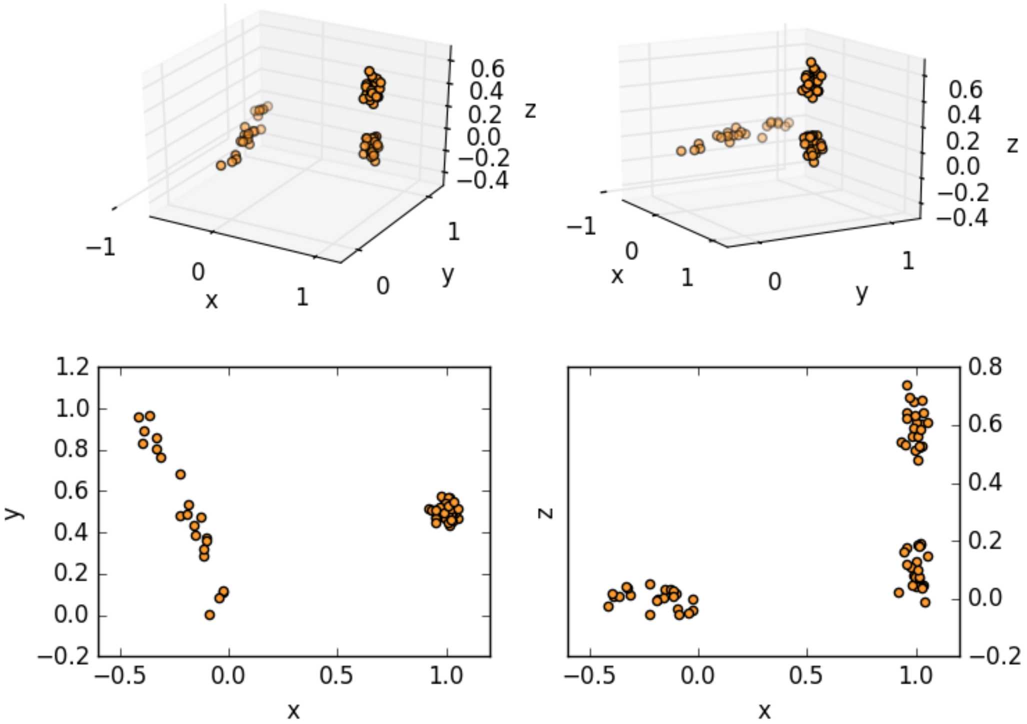


Figure 5.6: On the left the original cluster drawn in figure 3.4 was divided in two smaller ones. Now, by cutting the irrelevant dimension pointing downwards will make impossible to identify these two clusters.

The previous situations show that correlations among data points may depend on the locality of data. In probabilistic terms, assume the distribution is given by a mixture model

$$\rho(x) = \sum_{i=1}^{k} \pi_i \rho(x|i)$$

Each model $\rho(\cdot|i)$ has its specific conditional covariance $Cov[X|\ i\ ] \ = \ E[\ (X - \mu_i)^T(X - \mu_i) \mid i\ ] =: \Sigma_i$ where $\mu_i = E[\ X \mid i\ ]$, and associated to each one there is a PCA matrix $P_i^{(m)}$ whose rows are the first $m < d$ eigenvectors of $Cov[X|\ i\ ]$.

Assume Σ_i and μ_i, are parameters for each model, so that $\rho(x|i) = \rho(x|\Sigma_i, \mu_i, \theta_i)$ where θ_i is any other parameter not depending on the dimension d. If $\rho(\cdot|i)$ describes a good cluster, it should be well localized around μ_i, so that $\|\Sigma_i\|_F$ is not too big compared to others i's and $Cov[X]$, also, if its eigenvalues are similar in value, whatever projection we choose may catch well the characteristics of the cluster. If that is not the case, Σ_i may have very disparate (sparse) eigenvalues, and a (local) PCA would capture more precisely the relevant dimensions for this cluster locality. Complementary considerations are made in the appendix, where some affirmatives are made - though not rigorous, they shed light on when and why local PCA would be a nice escaping from the curse of dimensionality.

So, a possible 'cure' to the curse could be a local *principal component analysis* around dense data localities. There is a nice algorithm by Charu Aggarwal described in [2] which aims to implement analysis combining Clustering with local projections to principal components, the ORCLUS algorithm. In what follows we describe it. After we apply it and compare to the classic K-Means and K-Means + PCA schemes, with excellent results. In the appendix a python implementation is at disposal.

ORCLUS combines a PCA-like scheme while clusters in an hybrid iterative algorithm between K-Means and hierarchical clustering. It is designed to start with k_0 clusters and to gradually reduce their numbers until reach the desired k final clusters. The initial $k_0 > k$ centroids c_i are selected among the data set D, and every centroid (cluster representative) is always associated to a special reduced vector space V_i of relevant (i-th cluster) dimensions, whose dimension at iterate n is $l_{(n)}$. The initial vector spaces are the whole $\mathbb{R}^d$, and the algorithm gradually cuts down the dimensionality of these vector spaces V_i (which we shall call 'subspaces') towards the desired $l_{(n_{final})} = l$. At each iterate three steps are performed:

i. Assignment: projects every $x \in D$ and every c_i to the special subspace associated to the latter, then calculates the distance $d_{V_i}(x, c_i)$ in this dimension reduced space, where d_{V_i} is a given a choice of distance metric in V_i. Then we assign $x \in D$ to c_i if and only if $i = \text{argmin}_j \; d_{V_j}(x, c_j)$. Finally, the centroids are updated to the centers of the newly formed subsets $C_i = \{x \mid \text{argmin}_j \; d_{V_j}(x, c_j) = i\}$;

ii. Local PCA: Once D is partitioned in subsets C_i, a PCA is performed at each of them, and the special subspaces V_i are updated by extraction of the $l_{(n)}$ most relevant dimensions among C_i points. The n-th iteration lowers the previous dimension $l_{(n-1)}$ in such a way that the first iteration begins with $l_{(0)} = d$ (the full space $\mathbb{R}^d$) and the last finishes with $l_{(n_{final})} = l$. The exact proportions among $l_{(n-1)}$ and $l_{(n)}$ are discussed below;

iii. Merge: reduces the number of clusters from $k_{(n-1)}$ to $k \leq k_{(n)} < k_{(n-1)}$ by merging them one at a time according to the following routine.

It computes for every pair $i \neq j$, with $i = 1, 2, \cdots k_{(n-1)}$, the $l_{(n)}$ dimensional subspace V_{ij} associated to $C_i \cup C_j$ by performing a PCA, then computes the value of the objective function $r_{ij} = \sum_{C_i \cup C_j} d_{V_{ij}}(x, c_{ij})^2 / |C_i \cup C_j|$ where again $d_{V_{ij}}(\cdot, \cdot)$ is the distance function in the projected subspace V_{ij}.

Finally the pair for which r_{ij} is minimal is merged reducing the number of clusters by 1 and relabeling the indices so this routine can be repeated t times until $k_{(n-1)} - t = k_{(n)}$. When the last routine is performed (the desired $K_{(n)}$ is reached), the centroids are updated and newly calculated subspaces $V_{i'} = V_{ij}$ (after relabeling) are associated to the newly formed clusters $C_{i'} = C_i \cup C_j$ (the untouched V_i's for non-merged clusters are clearly maintained).

If $k_{(n)} = k$ the algorithm finishes. Else, we update the l and k values from $l_{(n)}$ to $l_{(n+1)}$ and $k_{(n)}$ to $k_{(n+1)}$, and iterate steps 1), 2) and 3).

The pseudocode is shown next. A Python implementation is shown in the appendix. It is a working code which amounts for excelent results as displayed in Figure 5.7, however, it should be enhanced by parallelizing its core procedure named 'merge'. In the figure (its visualization will be explained in detail later on), 96-dimensional vectors are represented as horizontal strips in a heat map. The colors intensities reflect the magnetude of each one of the 96 features. For now what is important is to verify the coherent patterns in each clustered group of data points; the robustness of detection amidst many noisy vectors (the lines breaking the beautiful patterns, specially in the clusters composed by 32 and 23 days); and the balance of the clusters sizes - there are no *overfitted* clusters with few points nor supersized clusters (which would reflect a method inability to distinguish well the patterns).

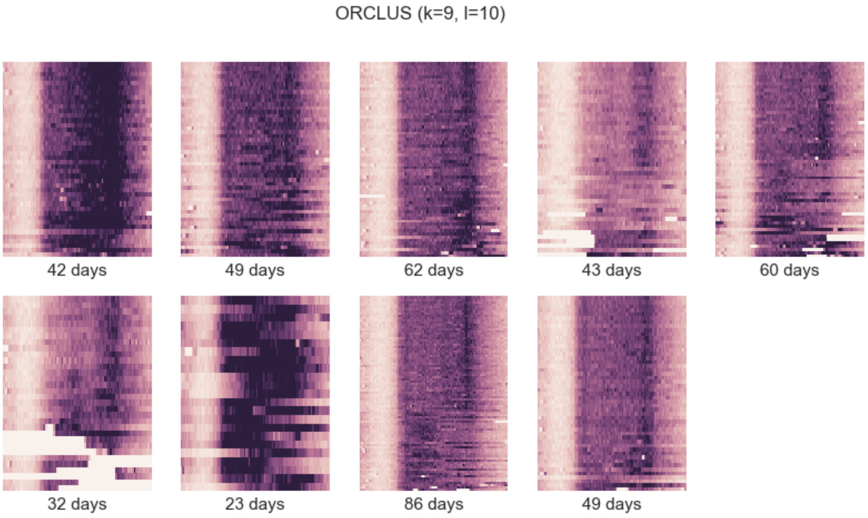


Figure 5.7: a sample of traffic counting data clustered by the ORCLUS algorithm. The days are displayed as 96-dimensional vectors along the horizontal axis, while the vertical axis are the different days. Although each cluster has a different number of clustered days, they have all the same height.

The parameter α is a reduction factor of the number of clusters at each iteration, so it should be between 0 and 1. Each iteration reduces the previous $k_{(n-1)}$ to $k_{(n)} = \max\{k, \lfloor \alpha\ k_{(n-1)} \rfloor\}$. A typical value for it is $\alpha = 0.5$. Associated to it there is the β parameter, the reduction factor of the subspaces dimensions so that $l_{(n)} = \max\{l, \beta\ l_{(n-1)} \rfloor\}$. β should be tuned so that the algorithm starts with $(k_{(0)}, l_{(0)}) = (k_0, l_0)$ and finishes at iteration n_{final} with $(k_{(n_{final})}, l_{(n_{final})}) = (k, l)$. Thus we must have

$$(k, l) = (\alpha^{n_{final}} k_0, \beta^{n_{final}} l_0)$$

$$\therefore \quad \frac{\ln(k/k0)}{\ln\alpha} = n_{final} = \frac{\ln(l/l_0)}{\ln\beta}$$

$$\Rightarrow \quad \beta = \mathrm{e}^{\ln\alpha \ln(l/l_0) \ln(k_0/k)}$$

The pseudocode has a main procedure, named ‘ORCLUS’, and the three associated procedures ‘Assign’, ‘FindVectors’, ‘Merge’, written as the three separate algorithms 5.6.1, 5.6.2, 5.6.3 and 5.6.4.

Algorithm 5.6.1: ORCLUS(k, l, k_0, d, α)

```
main

 $(c_1, c_2, \cdots c_{k_0}) \leftarrow$ *randomly selected*
 $k_c \leftarrow k_0$
 $l_c \leftarrow d$
 $\beta \leftarrow \mathrm{e}^{\ln\alpha \ln(l/l_0) \ln(k_0/k)}$

 for each $i \in \{1, 2, \cdots k_0\}$
   do $\mathrm{P}_i \leftarrow \mathrm{I}$

 comment: at each iteration, $k_{(n)} = k_c$ and $l_{(n)} = l_c$

 while $(k_c > k)$
   do
     $(C_1, C_2, \cdots C_{k_c}) \leftarrow \textsc{Assign}(c_1, c_2, \cdots c_{k_c}, P_1, P_2, \cdots P_{k_c})$

     for $i \leftarrow 0$ to $k_c$
       do $P_i \leftarrow \textsc{FindVectors}(C_i, l_c)$

     $k_{new} = \max\{k, \lfloor \alpha k_c \rfloor\}; l_{new} = \max\{l, \lfloor \beta l_c \rfloor\}$

     $(c_1, \cdots c_{k_{new}}, C_1, \cdots C_{k_{new}}, P_1, \cdots P_{k_{new}}) \leftarrow \textsc{Merge}(C_1, \cdots C_{k_c}, k_{new}, l_{new})$

     $k_c = k_{new};\ l_c = l_{new}$

 $(C_1, C_2, \cdots C_k) \leftarrow \textsc{Assign}(c_1, c_2, \cdots c_k, P_1, P_2, \cdots P_k)$

 return $(C_1, C_2, \cdots C_k)$
```

Algorithm 5.6.2: ASSIGN($c_1, c_2, \cdots c_{k_c}, P_1, P_2, \cdots P_{k_c}$)

```
for each $x \in D$
  do $ind(x) \leftarrow \underset{j=1,\cdots k_c}{\text{argmin}}\ d_{P_i}(x, c_j)$

comment: $d_{P_i}(\cdot,\cdot)$ is the distance function in the space projected by $P_i$

for $i \leftarrow 0$ to $k_c$
  do $C_i \leftarrow \{x \in D \mid ind(x) = i\}$

return $(C_1, C_2, \cdots C_{k_c})$
```

Algorithm 5.6.3: FINDVECTORS(C, l_c)

```
comment: $C$ is a cluster, $\mathbf{X}$ is a matrix whose rows are elements of C
  $\mathbf{X} \leftarrow C$

  $\mathbf{Cov}[\mathbf{X}] \leftarrow (\mathbf{X} - \mu)^T(\mathbf{X} - \mu)$

  $U, S, V \leftarrow svd(\mathbf{X})$

  $P \leftarrow [\ \vec{v}_1, \vec{v}_2, \cdots\ \vec{v}_{l_c}\ ]^T$

  comment: $\mu$ is the empirical mean, $P$ is the first $l_c$ rows of $V$

return $(P)$
```

Algorithm 5.6.4: Merge($C_1, \cdots C_{k_c}, k_{new}, l_{new}$)

```
for each $i < j \in \{1, 2, \cdots k_c\}$
    do
        $P_{ij} \leftarrow$ FindVectors($C_i \cup C_j, l_{new}$)
        $c_{ij} \leftarrow \mu(C_i \cup C_j)$
        $r_{ij} \leftarrow \sum_{x \in C_i \cup C_j} d_{V_{ij}}(x, c_{ij})^2 / |C_i \cup C_j|$
        comment: $\mu(C)$ is the cluster mean, $V_{ij}$ space associated to $P_{ij}$

while $k_c > k_{new}$
    do
        $(i', j') \leftarrow \operatorname{argmin}_{i<j}\{r_{ij}\}$
        $C_{i'} \leftarrow C_{i'} \cup C_{j'}$
        $c_{i'} \leftarrow c_{i'j'}$
        $P_{i'} \leftarrow P_{i'j'}$

        comment: discard $j'$ and relabel indices
        for $j \leftarrow j'$ to $k_c$
            do
                $j \leftarrow j - 1$
                $(i, j) \leftarrow (i, j - 1)$

        for each $j \neq i' \in \{1, 2, \cdots k_c - 1\}$
            do
                $P_{i'j} \leftarrow$ FindVectors($C_{i'} \cup C_j, l_{new}$)
                $c_{i'j} \leftarrow \mu(C_{i'} \cup C_j)$
                $r_{i'j} \leftarrow \sum_{x \in C_{i'} \cup C_j} d_{V_{i'j}}(x, c_{i'j})^2 / |C_{i'} \cup C_j|$

        $k_c \leftarrow k_c - 1$

return $(c_1, \cdots c_{k_{new}}, C_1, \cdots C_{k_{new}}, P_1, \cdots P_{k_{new}})$
```

5.7 An Application

In what follows it is presented an application of the high dimensional clustering tools discussed in this chapter for the detection of patterns in traffic counting data from the *Departamento Nacional de Infraestrutura de Transporte* (*DNIT*). The data set is a temporal series like in Figure 5.8 where each day is divided in 96 intervals of 15 minutes.

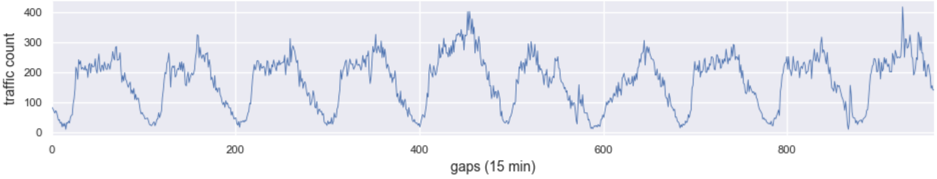


Figure 5.8: A typical timeseries of the *DNIT* data set.

In order to cluster this series into groups of days, we divide the long time series into small series representing a 24 hours day, divided in 96 intervals of 15 minutes. So, each day in the series is represented by a 96-dimensional vector $(c_1, c_2, \cdots c_{96}) \in \mathbb{N}_*^{96}$. We will visualize the clusters data through heat maps, an example of which is given in Figure 5.9.

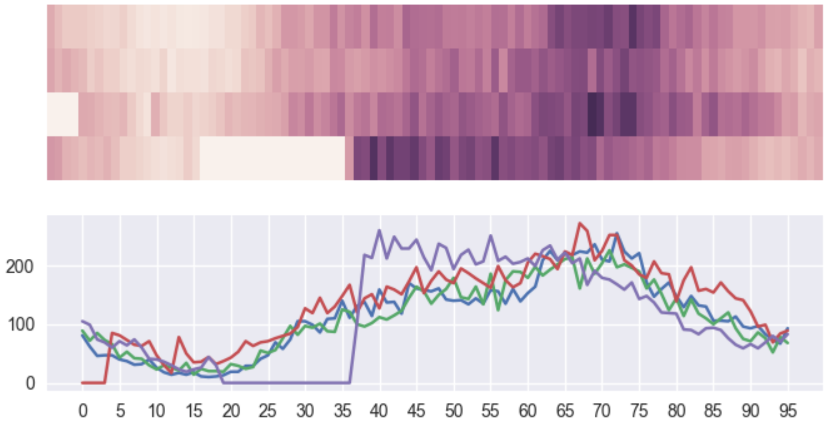


Figure 5.9: Four days were picked and displayed as a heat map above, and below is displayed as a time series of a 96-dimensional vector. The lighter the color, the closer to zero is the traffic counting, the darker the color, the higher the count of cars passing by during the corresponding 15 minute interval. Notice the series (in purple) with a bunch of zero countings between the 17^{th} and the 36^{th} intervals.

Each interval of fifteen minutes has a color, the lighter it is, the closer to zero is the counting, while the darker it is, the bigger the counting is. The dimension $d = 96$ can be considered to be high, but let's first apply the K-Means, in Figure 5.10, to start digging some insights.

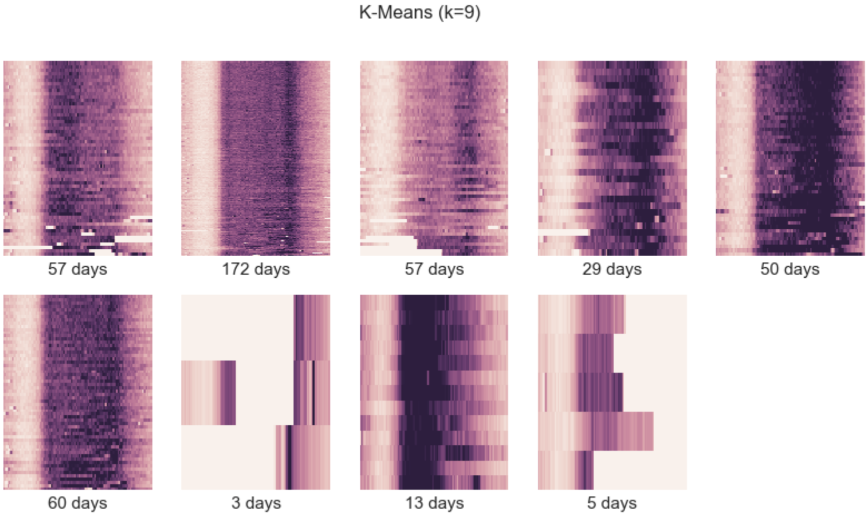


Figure 5.10: K-Means clustering with $k = 9$

Later on we will discuss why $k = 9$ was a good choice of parameter. K-Means performs poorly because it catches two clusters with too few points (3,5 and 13 days respectively) while it forms clusters with too many data points (the one with 172 days). The former observation is an ev-

idence of the *overfitting* problem: in high dimensions, it is easy to find some small groups of points quite far from all the others - the so-called *outliers*. Now the latter observation is an evidence of the inherent sparsity of data points in high dimensions: more or less all the points are equidistant among themselves thus making it difficult to differentiate the data points, so a bad clustering procedure will tend to put a good part of the data under one big cluster.
We apply the K-Means + PCA scheme to this same set of data points, the result given in Figure 5.11. It does not perform really better than the K-Means. Still there is a big cluster of 166 days,two small clusters with 13 days and one last with only 4 days. This means that probably the good clusters have their own special directions (as explained in the last chapter), so that a global PCA is not enough to reduce dimensionality without losing essential descriptive data. We appel to the ORCLUS technique in Figure 5.12. We notice that (*i*) the ORCLUS does not capture clusters with few isolated points; (*ii*) it forms well balanced clusters, the bigger one possessing 86 days. This means that the ORCLUS is able to detect subtle peculiarities of the high dimensional vectors thus not forming big clusters, and it is robust against *outliers* which naturaly occurs when the dimension is high, meaning that it will not isolate them in small clusters.

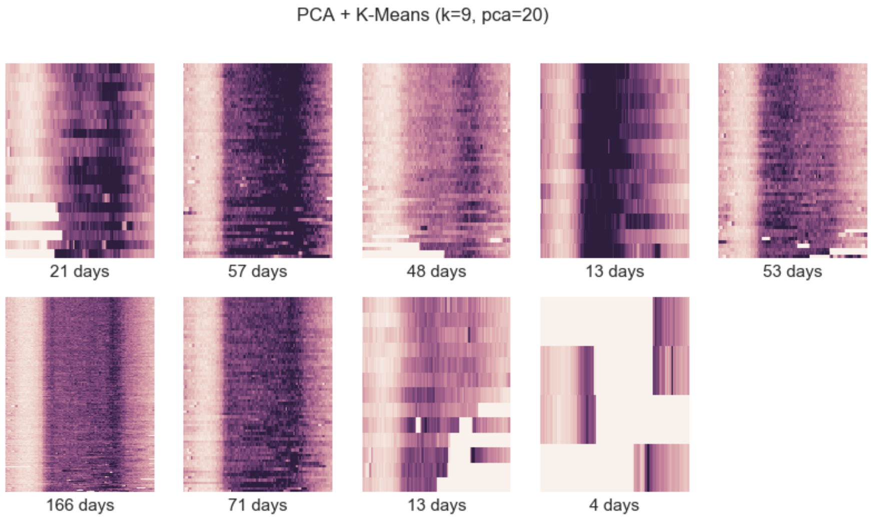


Figure 5.11: K-Means performed after a reduction of dimension by Principal Component Analysis.

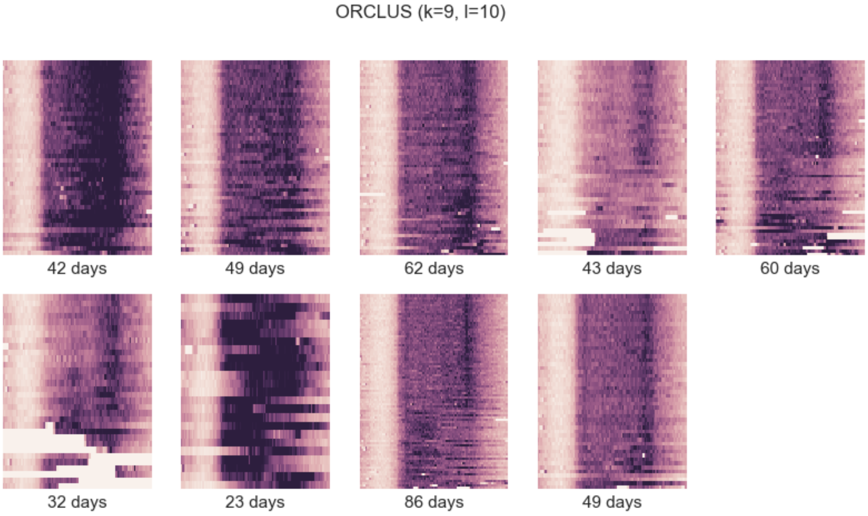


Figure 5.12: ORCLUS technique. Notice that the clusters' sizes are balanced.

We show another clustering performed over a new data set of the same kind. We only display the commented Figures 5.13, 5.14 and 5.15.

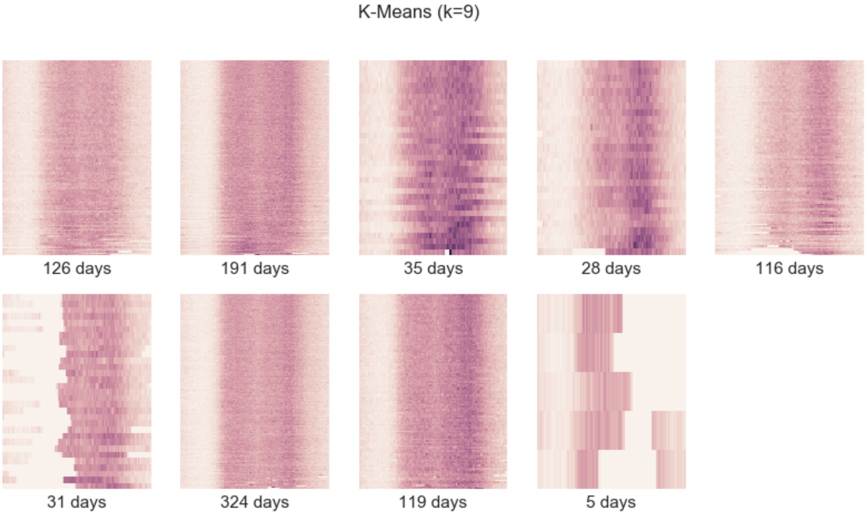


Figure 5.13: K-Means. Notice one very small cluster with only 5 days and others three with around 30 days. By contrast a huge cluster composed of 324 days, comprising almost one third of the data, which sounds awkward. But a quick inspection make us believe the big cluster is reliable - indeed, when we have so many features to distinguish between two vectors, we lose the notion of what 'to be different' means - this inability to distinguish well the data points is just a manifestation of the *curse of dimensionality*.

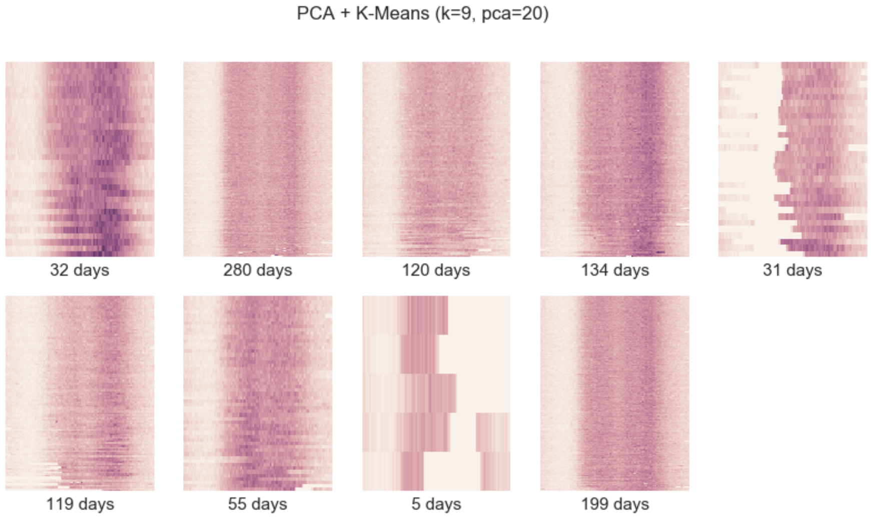


Figure 5.14: K-Means performed after a reduction of dimension by Principal Component Analysis. We notice a better balance in the clusters sizes though it persists a very small cluster and a huge one with 280 days (but less than 324), as well a second huge cluster with 199 days.

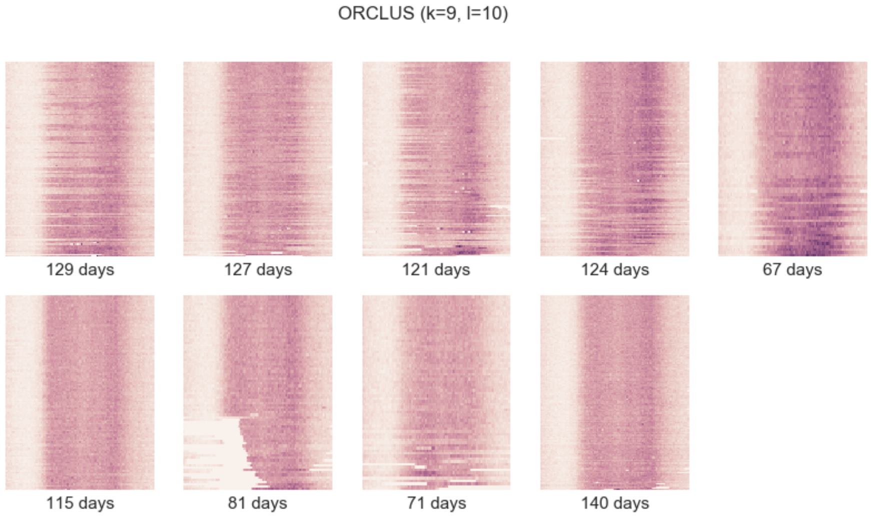


Figure 5.15: ORCLUS. We can see balanced clusters sizes and cluster with an appealing aspect of a certain coherence of their shapes. Notice specially the cluster with 81days. The ORCLUS was able to group many vectors with missing data together with clean data points, and a quick visual inspection makes clear that the group is coherent. Notice also that the clusters seems to be more 'noisy'. The ORCLUS identifies subtle differences among high dimensional vectors in the relevant dimensions, but it could well be that in the remaining subdmensions, the vectors diverge by a huge difference, so we see this 'noisy' aspect. In the previous schemes, these 'noisy' points probably were all collected in a sole big cluster.

Once we cluster the data, we can study the clusters' profiles, like in Figure 5.16. Grouping the points gives us insights from the data set, uncover patterns from which we could take decisions, or even try to predict events, or can make us improve statistics calculations. For the former application, take as example the mentioned figure. The first cluster has mean countings close to zero - probably this cluster caught the majority of null counting *outliers* - while the next two clusters show similar mean counting but have different variances - the second cluster shows greater variability of counting profiles, the third presents a more uniform aspect. Have not we clustered the data set before analysing these statistics then we would have mixed the different profiles of days in a single non-informative mean-variance statistics.

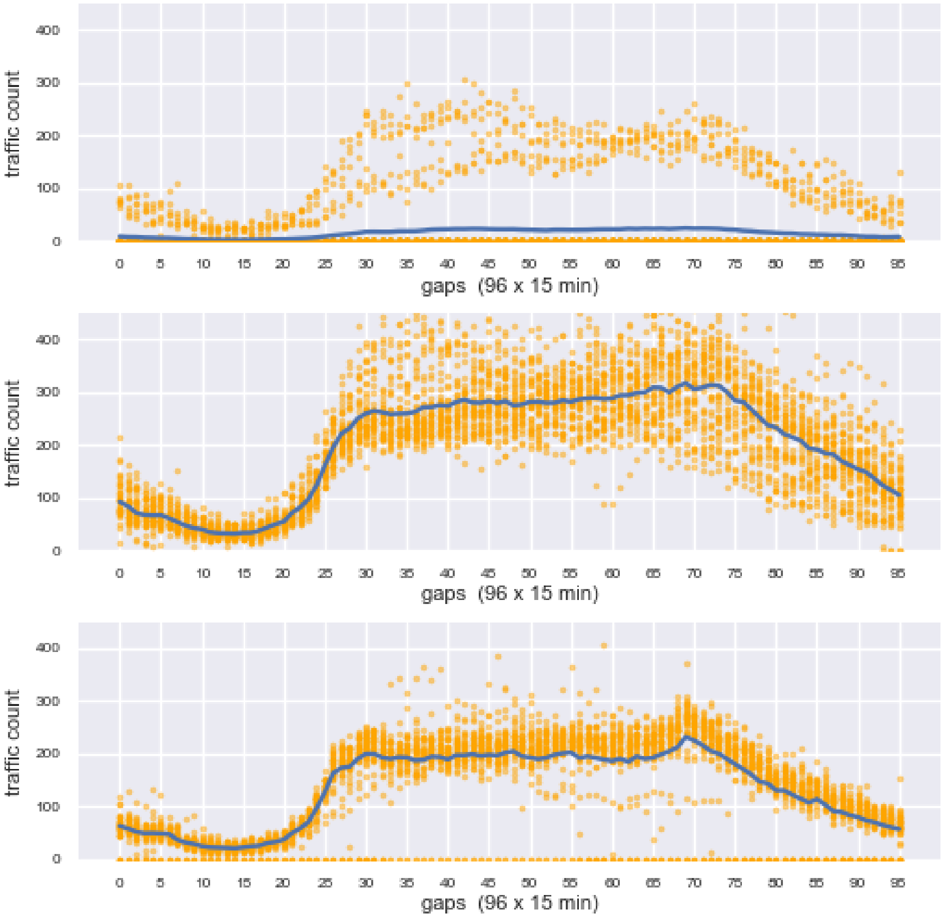


Figure 5.16: We see the profile of three typical clusters. The blue line represents the median of the counts for each 15 minutes interval, *i.e.*, it represents the typical traffic count profile for days belonging to such cluster. The median of each cluster was plotted together with actual realizations of that cluster, displayed in yellow dots. Although these realizations were not distinguished among themselves, they give an idea of the typical statistical fluctuations one can see inside each cluster. For example, in the first we can see many yellow dots marking null countings, so many of them that they pushed down the median statistics towards 0; in the second cluster, we notice a huge variance around the median statistics.

5.8 Validating the results

In order to cluster the data, we need to specify some parameters. In the former application, these parameters were the predefined number of clusters, k, for both ORCLUS and K-Means; the reduced dimension of the *principal component analysis*, denoted here as *dim*; the subdimension l associated to the PROCLUS projections over the relevant dimensions around the data cluster localities. We need some criteria to tune these parameters, and tunning them well will result in better clustering performance and clusters quality.

Ideally, two clusters should be well *separated* groups points in space - otherwise we could not distinguish them clearly thus making no sense declaring them as two *different* clusters instead of one single bigger cluster. Also, *inside* the cluster, we would expect the variability not to be so high, otherwise, the points would be so different among themselves that it would have no sense in declaring them *similar.* So these two properties, *separation* and '*intra-similarity*' are key to identify quantitatively what a good cluster should be. A *cluster validation index* is defined as a quantity measured over the data which quantifies somehow these characteristics, and give a scale of how far or how close we are from ideal clustering configurations.

In the *probabilistic framework* we represent the underlying generation mechanism of the data by a *pdf* $\rho = \rho(x)$, so that each data point is the realization of a variable $X_i \colon \Omega \to \mathbb{R}^d$ distributed according with $X_i \sim \rho(x)$. Then a *hard clustering* that classifies n realizations $X_1, X_2, \cdots X_n$ into any one of the labels belonging to the *target space* $\mathcal{Y}$ can be represented by a function of the realizations

$$C^{(n)} \equiv C_\rho(X_1, \cdots, X_n)$$

Defined like

$$\begin{array}{rccc} C^{(n)}\colon & \Omega & \to & \mathcal{Y}^n \\ & \omega & \mapsto & C^{(n)}(\omega) \end{array}$$

An *index score* for n data realizations is a function

$$\begin{array}{rccc} I^{(n)}\colon & \mathbb{R}^{d\times n} \times \mathcal{Y}^n & \to & \mathbb{R} \\ & (\mathbf{X}, \mathbf{y}) & \mapsto & I^{(n)}(\mathbf{X}, \mathbf{y}) \end{array}$$

Where $\mathbf{X} = (x_1, x_2, \cdots x_n) \in \mathbb{R}^{d\times n}$ is the matrix of data realizations and $\mathbf{y} = (y_1, y_2, \cdots y_n) \in \mathcal{Y}^n$ is the vector of labels in the target space $\mathcal{Y}$, (this notation eludes the framework of *supervised learning* discussed in Chapter 1).

An index score measures how 'good' the label assignments y_i given to every realization x_i are, in the sense of producing very compact and well separated groups of points in $\mathbb{R}^d$. We combine $I^{(n)}$ and $C^{(n)}$ in a composite-like function

$$I^{(n)} \circ C^{(n)} = I^{(n)}(X_1, X_2, \cdots, X_n; C_\rho(X_1, X_2, \cdots, X_n))$$

We call the function $R^{(n)} := I^{(n)} \circ C^{(n)}$ the *index validity* of the clustering $C^{(n)}$ associated to $I^{(n)}$. For the validation of our results we use the following validation indices:

SSB:

$$R^{(n)}_{SSB}(X_1, X_2, \cdots X_n) = \sum_{i=1}^{n} \|X_i - \mathbf{m}_{y_i}\|^2$$

where $y_i \in \{1, 2, \cdots k\}$ is the X_i assignment, $\mathbf{m}_j$ is the i^{th} cluster center. Notice that $\mathbf{m}_j = (X_1, \cdots X_n)$, the limit value of a cluster algorithm.

SSW:

$$R^{(n)}_{SSW}(X_1, X_2, \cdots X_n) = \sum_{j=1}^{k} n_j \|c_j - \bar{X}\|^2$$

where $n_i = |C_i|$ is the cluster size and $\hat{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$.

Calinski-Harabasz:

$$R^{(n)}_{SSW}\frac{R_{CH}/(n-1)}{R_{SSW}/(n-k)}$$

(we omitted the '(n)')

Dunn:

$$R^{(n)}_{Dunn} = \frac{\min\limits_{j=1,\cdots k}\left\{\min\limits_{j' \neq j} d(\mathbf{m}_j, \mathbf{m}_{j'})\right\}}{\max\limits_{j=1,\cdots k} diam(C_j)}$$

Davies-Bouldin:

$$R^{(n)}_{DB} = \frac{1}{k}\sum_{j=1}^{k} R_j$$

where $R_i = \max_{j \neq j'} \frac{S_j + S_{j'}}{d(\mathbf{m}_j, \mathbf{m}_{j'})}$,
and $S_j = \frac{1}{|C_j|} \sum\limits_{X_i \in C_j} d(X_i, \mathbf{m}_j)$

C index:

$$R^{(n)}_{C} = \frac{S - S_{min}}{S_{max} - S_{min}}$$

Where $S = \sum\limits_{j=1}^{k} \sum\limits_{x \neq x' \in C_j} d(x, x')$; $S_{min} = \sum\limits_{s=1}^{N_W} d_s$; $S_{min} = \sum\limits_{s=n-N_W}^{n} d_s$
with $\{\mathrm{d}_1 < d_2 < \cdots d_{n-1} < d_n\} = \{d(x, x') \mid x, x' \in \cup_j C_j\} and \mathrm{N}_W = \sum\limits_{j=1}^{k} \frac{n_j(n_j - 1)}{2}$

Silhouette:

$$R^{(n)}_{S} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{b(i) - a(i)}{\max\{a(i), b(i)\}}\right)$$

Where $a(i) = \frac{1}{n_j - 1}\left(\sum\limits_{x \in C_j/\{x_i\}} \|x - x_i\|^2\right)$ is the *intra-cluster* mean distance to $x_i \in C_j$ and $b(i) = \min\limits_{l \neq j}\left\{\frac{1}{|C_l|}\left(\sum_{x \in C_l} \|x - x_i\|^2\right)\right\}$ is the minimum *inter-cluster* mean distance from $x_i \in C_j$ to C_l, $l \neq j$

For interpretations of the geometric meaning of these indices, see [38],[12]. These indices must all be as high as possible to indicate a good clustering configuration, with the exception of the Davies-Bouldin index, which should be ideally as biggest as possible.

In the following we test three of these indices with four clustering configurations (they were clustered in order to calculate the indices but the clustering result is not displayed to avoid visual pollution), the first is at the top left, and it was modified in order to: (*i*) increase the separation at the bottom left; (*ii*) decrease the cluster radius in the top right; (*iii*) both decrease the radius and augment the separation. The remaining indices behave similarly as shown below.

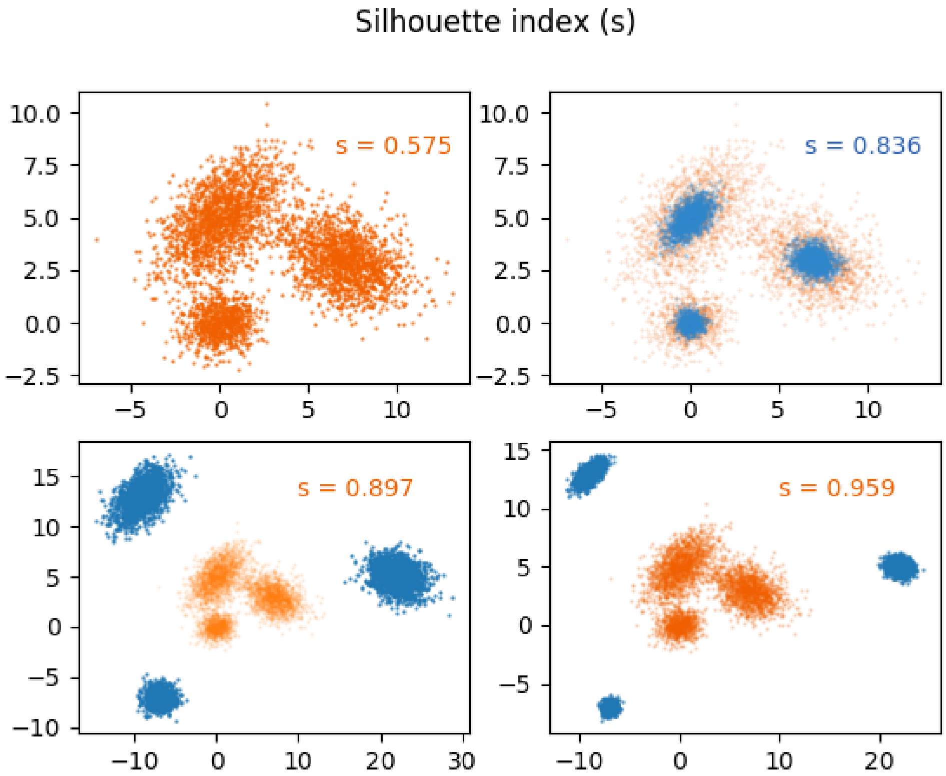


Figure 5.17:

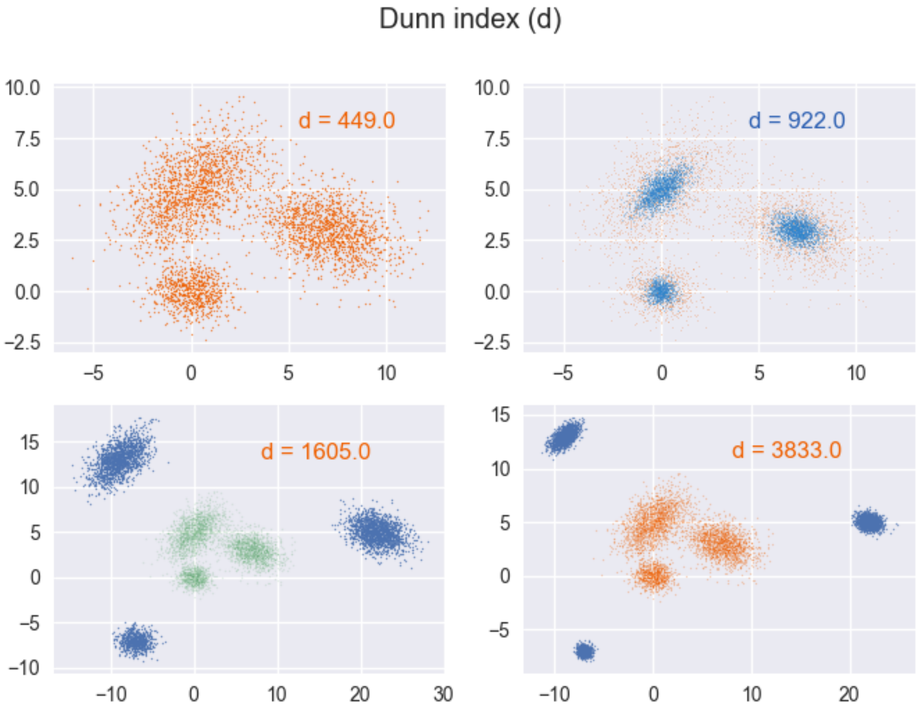

Dunn index (d)
d = 449.0
d = 922.0
d = 1605.0
d = 3833.0
10.0
7.5
5.0
2.5
0.0
−2.5
−5
0
5
10
15
10
5
0
−5
−10
−10
0
10
20
30

Figure 5.18:

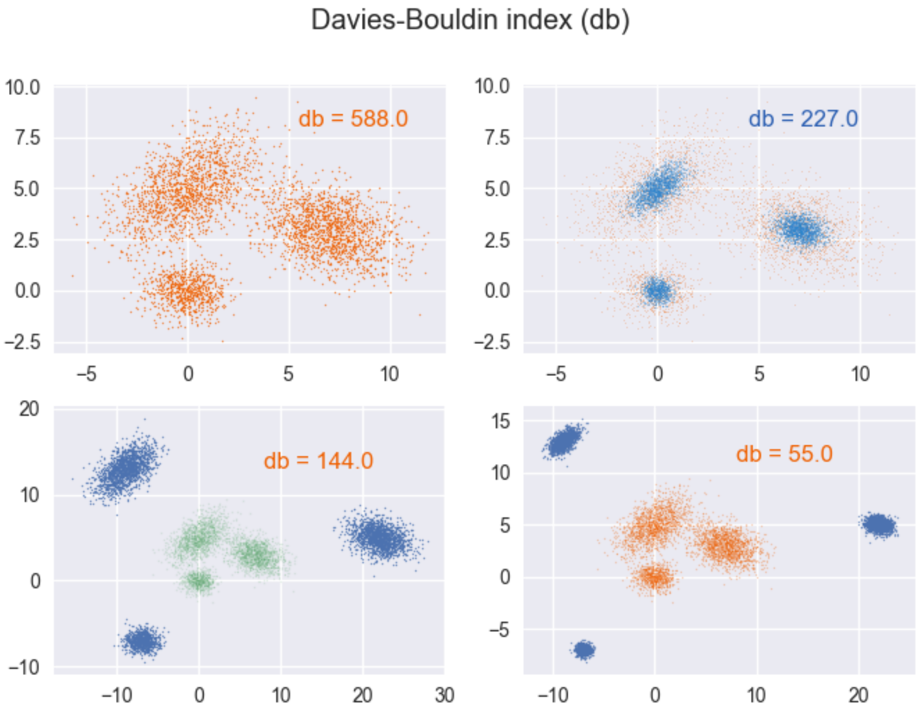


Figure 5.19:

Now we turn to the results obtained in the last section. They were all taken with $k = 8$ and PCA dimension around 20. We selected these numbers by checking some cluster validity indices for varying parameters values. For the ORCLUS, as it costs much to perform, we fixed a subdimension $l = 10$, after some tentatives which showed that augmenting l costs too much for small enhancements. So we varied only k from 7 to 10. We display the results for the KMeans+PCA scheme in Figure 5.20.

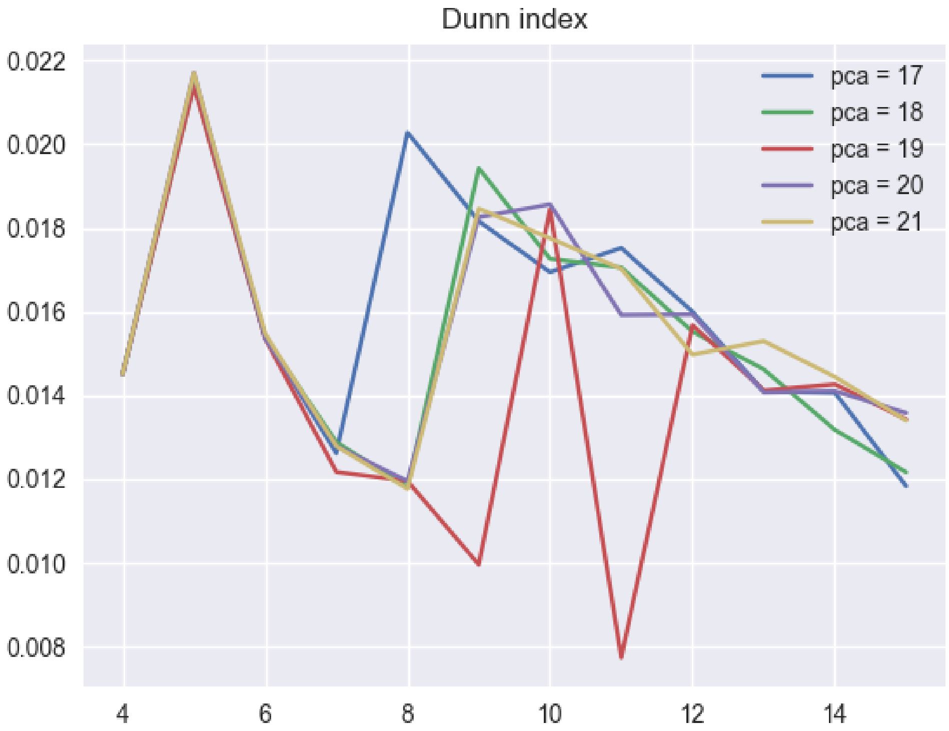


Figure 5.20:

The Dunn index should ideally be as biggest as possible. For the K-Means + PCA scheme it tells that the ideal cluster configuration is for $k = 5$. Notice that it is independent of the dimension of the PCA reduction, indicated in the legend as 'pca'. But for the data used here it sounds unrealistic, because each point is a day, we we would expect that something around 7 is reliable, one for each day week day. It seems more informative to analyse the other pics. We see that me most expressive ones are around $k = 8, 9, 10$. We notice that the blue line seems to be special, but in fact all these lines behave similarly, which sounds like evidence that we have some freedom in adjusting the PCA dimension pca. Indeed we can check that trying broader intervals for pca doens't affect the overall aspect of the clustering.

Let's check now the Silhouette index, Figure 5.21, which should be ideally as big as possible. We see descending curves, a common pic at $k = 5$, which we ignore, and reliable pics running between $k = 8$ to $k = 10$, specially for $pca = 17, 19, 21$. The yellow curve $pca = 21$ seems very special, and indicates a good clustering at $k = 8$.

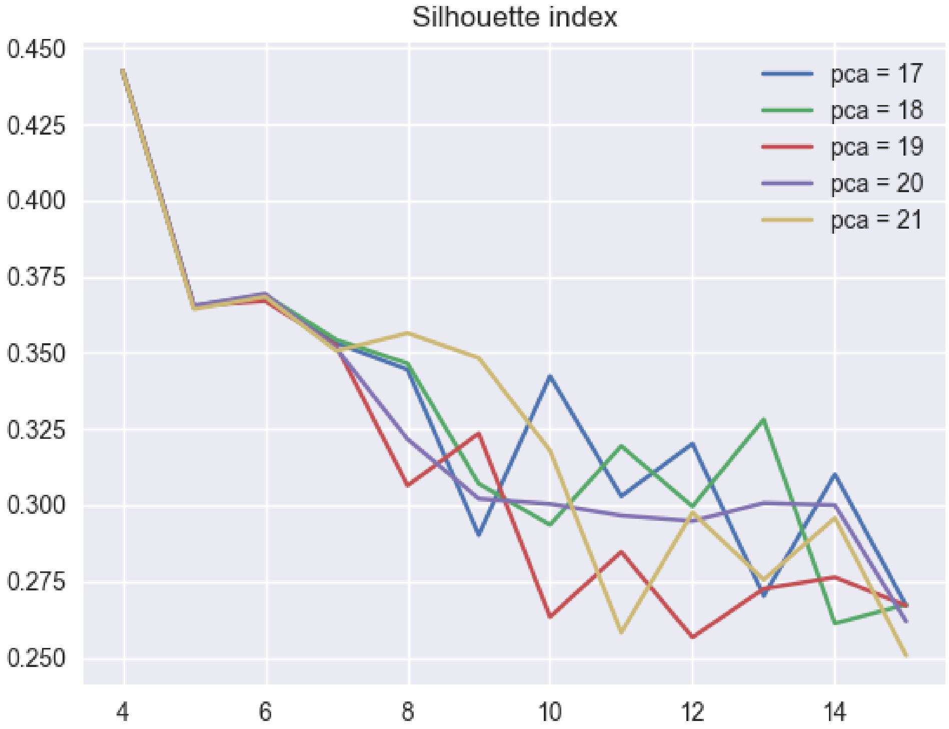


Figure 5.21:

The Calinski-Harabaz index, in Figure 5.22 gave very characteristic results: highly similar descending curves. Ideally, this index should be as biggest as possible, therefore, it tells us to chose the smaller k possible, and that the choice for *pca* does not affect the overall clustering result.

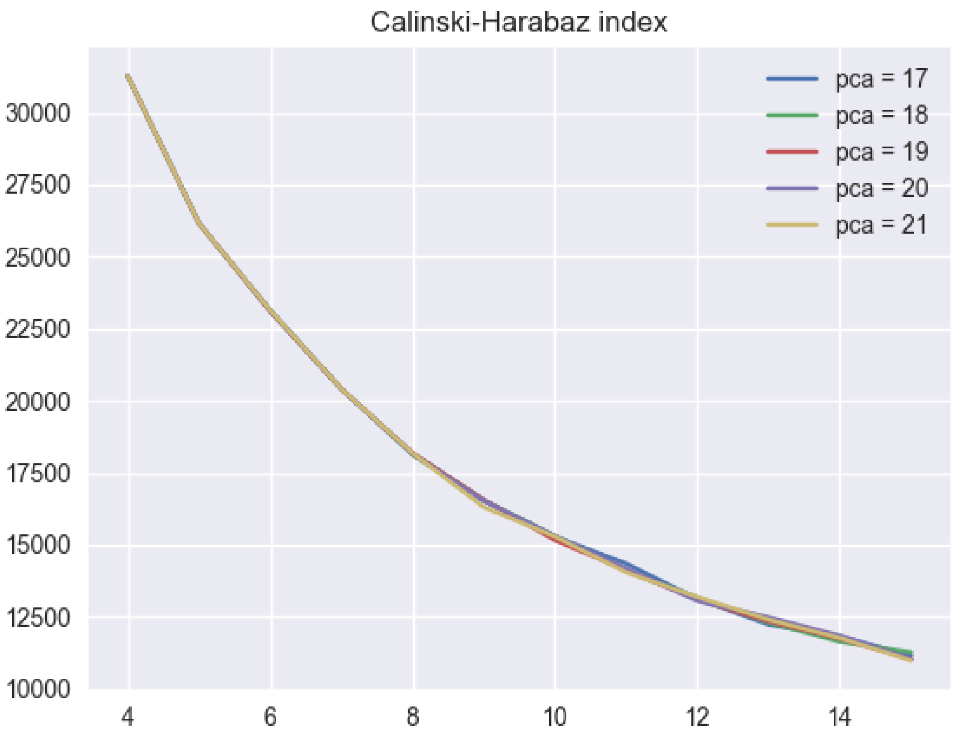
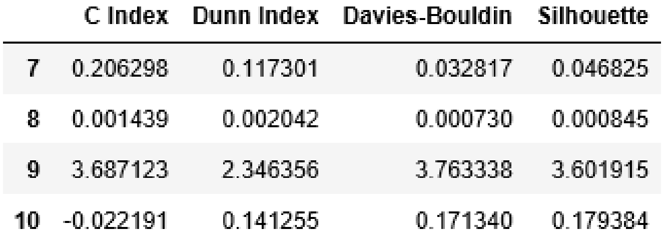


Figure 5.22:

Hence we can trust that the chosen parameters $k = 8$ and $pca = 20$ used in the previous images are good parameters.

Now we present a table of tests performed for the ORCLUS. It is a heavy algorithm, so we tried only a small range for k, specially because the previous empirical calculations gave us $k = 8$.



	C Index	Dunn Index	Davies-Bouldin	Silhouette
7	0.206298	0.117301	0.032817	0.046825
8	0.001439	0.002042	0.000730	0.000845
9	3.687123	2.346356	3.763338	3.601915
10	-0.022191	0.141255	0.171340	0.179384

Figure 5.23:

Looking the table we check that the bigger results are for $k = 9$. Also, we notice the similar behaviour among all the indices. Its an interesting information, and it should be emphasized that the ORCLUS is able to catch more underlying important features of the data than the other methods, thus resulting in better validation evaluations. Therefore we perform a last comparative clustering with $k = 9$, $l = 10$, $pca = 20$.

Conclusions

The main goal of this work was to put on solid grounds the mathematical understanding of the problem of *overfitting* and the *Curse of Dimensionality* phenomenon. We had as field test the data set provided by the *Departamento Nacional de Infraestrutura e Transportes*, which was composed of 96-dimensional vectors representing traffic counting data over a day. We has also the desire to offer an extensive review of some classical methods in Cluster Analysis, namely, the K-Means which was studied in depth; the soft K-Means, which was understood in a Mixture Model context; and the Gaussian Mixture Model itself.

These objectives were achieved: we developed an understanding of Supervised Learning from a probabilistic point of view; we analysed in great detail the Mixture Model and its applications; we made rigorous (by depicting the article by Ismael and Selim) the proof of convergence of the hard K-Means in the context of Discrete Optimization; We developed insights about the *overfitting* problem and how it naturally occurs in high-dimensional regimes; thus we tried to ilustrate both mathematically and empirically the *Curse of Dimensionality* in the context of Concentration of Measure; and implemented, tested and validated the ORCLUS algorithm by Charu Aggarwal, in an attempt to fight the curse of dimensinality while avoiding the overfitting problem.

The field of Unsupervised Learning lies in the frontier of recent the developments in Machine Learning. Therefore it is natural to have a multitude of techniques being developed without a proper mathematical analysis background, and a myriad of articles with important results are being produced, most of them relying on heuristics assumptions - the author stress his belief that paving the way to clear rigorous mathematics is essential to the maintain this development by giving insights on new techniques, by bringing to the surface old results often ignored (like why the hard K-Means converge?) and by creating a pool of accessible scientific material which allows anyone interested in Machine Learning to participate in this promising field.

Still there is much to discover, and specially, to *implement*, as every day the volume of complex data grows astonishingly. The typical dimensionality of data generated by our smartphones, computers, sensors, cameras, and alike is only growing in a fast pace, so one cannot expect all this data will be *supervised*, and even if it does, there is much information hidden inside all these bytes which Supervised Learning techniques will uncover!

Appendix A

K-Means Python implementation

A.1 Hard K-Means

```python
class KMeans():

  def __init__(self,X,k,c=None,max_iter=80,sel):
    # X = n_samples x n_features numpy array,
    # X = [x1,x2,x3, ... ] , xi = [xi1,xi2, ... ]
    # c = k numpy array of centroids, c = [c1,c2, ... , ck]
    # max_iter limits the maximum number of iterations
    self.X = X
    self.k = k
    self.d = X.shape[1]
    self.n = X.shape[0]
    self.max_iter = max_iter

  def kmeans(self,c=None):

    self.it=0 #iteration number
    # randomly selects initial centroids, chosen among X values
    if c is None:
      self.c = X[random.sample(range(self.n),self.k)]
    else:
      self.c = c

    # the first iteration we start outside the loop
    self.it += 1
    # in order to calculate all the distances in a single
    #operation, we stack k copies of X
    S = np.vstack([self.X]*self.k)
    # computes the difference (x_i - c_j) for each x_i in X,
    # for every centroid c_j, and stacks everything in a big
    # d x k x n matrix, then computation of distances is
    # optimally performed and finally we arrive at k x n matrix
    # from which we select the arguments where $|xi - cj|$
    # is minimal, resulting in the labels' vector named (self.)'colors'

    l = []
    # computes label assignments
    for i in range(self.d):
      #stacks n copies of the i-th centroid to
      #subtract it from each xi in X
      C = np.outer(np.ones(self.n),self.c[:,i])
      x = C.T.flatten() - S[:,i]
      l.append(x.reshape(x.shape[0],1))
    D = np.array(l).reshape(self.d,self.k,self.n)
    distances = np.linalg.norm(D,axis=0)
    #colors = clusters' labels
    self.colors = np.argmin(distances,axis=0)
    #update centroids
    self.c_new = np.array([self.X[self.colors==i].mean(axis=0) for i in range(self.k)])

    # we enter the while loop. 'np.allclose' evaluates if the new
    # centroids are close within a specified tolerance e (\epsilon).
    # Here, it is automatically pre-setted by numpy by the value e = 1e-5
    while (np.allclose(self.c,self.c_new)==False)&(self.it<self.max_iter):
      self.c=copy.deepcopy(self.c_new)
      self.it += 1
      S = np.vstack([self.X]*self.k)
      l = []
      for i in range(self.d):
        C = np.ones(self.n),self.c[:,i])
        x = np.outer(C.T.flatten() - S[:,i]
        l.append(x.reshape(x.shape[0],1))
      D = np.array(l).reshape(self.d,self.k,self.n)
      distances = np.linalg.norm(D,axis=0)
      #colors = clusters' labels
      self.colors = np.argmin(distances,axis=0)
      self.c_new = np.array([self.X[self.colors==i].mean(axis=0) for i in range(self.k)])
```

A.2 Soft K-Means

```
class SoftKMeans():

  def __init__(self,X,k,max_iter=80):
    # X = n_samples x n_features numpy array, X = [x1,x2,x3, ... ], xi = [xi1,xi2, ... ]
    # c = centroids
    self.X = X
    self.k = k
    self.d = X.shape[1]
    self.n = X.shape[0]
    self.max_iter = max_iter

  def softKmeans(self,b,c=None):

    self.b = float(b)
    self.it = 0 #iteration number
    if c is None:
      self.c = X[random.sample(range(self.n),self.k)]
    else:
      self.c = c

    self.it += 1

    S = np.vstack([self.X]*self.k)
    l = []
    for i in range(self.d):
      x = np.outer(np.ones(self.n),self.c[:,i]).T.flatten() - S[:,i]
      l.append(x.reshape(x.shape[0],1))
    distances = np.linalg.norm(np.array(l).reshape(self.d,self.k,self.n),axis=0)
    z = np.exp(-b*distances)
    self.colors = (z/z.sum(axis=0)).T
    self.c_new  = (X.T.dot(self.colors)/self.colors.sum(axis=0)).T

    while (np.allclose(self.c,self.c_new)==False)&(self.it<self.max_iter):
      self.it += 1
      self.c = copy.deepcopy(self.c_new)

      S = np.vstack([self.X]*self.k)
      l = []
      for i in range(self.d):
        x = np.outer(np.ones(self.n),self.c[:,i]).T.flatten() - S[:,i]
        l.append(x.reshape(x.shape[0],1))
      distances = np.linalg.norm(np.array(l).reshape(self.d,self.k,self.n),axis=0)
      z = np.exp(-b*distances)
      self.colors = (z/z.sum(axis=0)).T
      self.c_new = (X.T.dot(self.colors)/self.colors.sum(axis=0)).T

    self.means = copy.deepcopy(self.c_new)
    self.covariances = np.array([np.diag((.1/self.b)*np.ones(self.
   d)) for i in range(self.k)])

  def softProbs(self,Y):

    n = Y.shape[0]
    S = np.vstack([Y]*self.k)
    l = []
    for i in range(self.d):
      x = np.outer(np.ones(n),self.c[:,i]).T.flatten() - S[:,i]
      l.append(x.reshape(x.shape[0],1))
    distances = np.linalg.norm(np.array(l).reshape(self.d,self.k,n
   ),axis=0)
    z = np.exp(-self.b*distances)

    return (z/z.sum(axis=0)).T
```

A.2.1 Zangwill's Global Convergence Theorem

So far we have only studied in depth the convergence of the hard K-Means and a generalization for different dissimilarity matrices. For most algorithms, the exact convergence for an optimal solution is difficult to guarantee - even for the simple cases when $D(\cdot,\cdot)$ is the squared euclidean norm or the L^p norm, some involved results were needed.
But often an algorithm converging towards a partial optimal solution or something similar to it is the best one can have, and sometimes even that returns effective results. For these situations, there is a very general result by Zangwill [37],[6],[14], which states a general convergence proof (in some special sense) for classes of algorithms characterized by the so-called *point-to-set mapping*. A concise overview of this very interesting result is given in [**?**].
Zangwill's argument generalizes those arguments used in the convergence theorems for the algorithms highlighted in the previous section and of section 2.?, also, it will appear once again in the *Fuzzy K-Means* convergence theorem by Bezdek [**?**]. So, by its importance and constant use here, we'd better state it in all its generality.
Zangwill's theory involves the concept of *point-to-set mapping*, or a set-valued function:

Definition A.2.1. *Given two sets, X and Y, a point-to-set mapping from X to Y is a set-valued mapping $\Phi\colon X \to \mathcal{P}(Y)$ which assigns to each $x \in X$ a subset $\Phi(x) \subset Y$, i.e., an element of the power set of Y.*

One can see this as a generalization of the notion of a function from X to Y, as one such function $\phi\colon X \to Y$ could be redefined as a point-to-set mapping $\Phi(x) = \{\phi(x)\}$.
Let's begin our analysis with a general definition of an *iterative algorithm.*

Definition A.2.2. *Let X be a set and $x_0 \in X$. An iterative algorithm $\mathcal{A}$ with initial point x_0 is a point-to-set mapping $\mathcal{A}\colon X \to \mathcal{P}(X)$ associated to the family of iterative sequences $\{x_n\}_{n=0,1,2,\cdots}$ starting at x_0 and generated according to*

$$x_{n+1} \in \mathcal{A}(x_n)$$

Remark. *Each iterative sequence $\{x_n\}_{n=0,1,2,\cdots}$ is the result of a particular method for the implementation of the iterative algorithm $\mathcal{A}$. Hence $\mathcal{A}$ represents a class of algorithms or implementations.*

Take as example the function φ defined in section 2.?. Clearly the definition of φ depends on the data set $D = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n\}$, so let's denote φ by the new notation $\varphi = \varphi_{\{\mathbf{x}_1,\mathbf{x}_2,\cdots\mathbf{x}_n\}}(\mathbf{M})$.
Now one can define for each $\mathbf{M} = (\mathbf{m}_1, \mathbf{m}_2, \cdots \mathbf{m}_n)$ the set

$$\mathcal{A}(\mathbf{M}) = \{\varphi_{\{\mathbf{x}_1,\mathbf{x}_2,\cdots\mathbf{x}_n\}}(\mathbf{M}) \mid \mathbf{x}_i \in \mathbb{R}^{\mathrm{d}} \text{ and } n \in \mathbb{N}\}$$

i.e., it is the set of possible values for $\varphi(\mathbf{M})$ given all configurations for the data sets D and $n = \#D$.
Now we define the notion of a *descent function*:

Definition A.2.3. *Given a subset $\Gamma \subset X$ and an iterative algorithm $\mathcal{A}$, a continuous real-valued function $E\colon X \to \mathbf{R}$ is called a descent function if it satisfies:*

i. if $x \notin \Gamma$ and $y \in \mathcal{A}(x)$, $E(y) < E(x)$

ii. if $x \in \Gamma$ and $y \in \mathcal{A}(x)$, $E(y) \leq E(x)$

We call Γ the *solution set*, because in practice it represents a set of solutions we aim to reach through the iterations of the algorithm. The idea behind the definition of a descent function is to have an energy function such that it always decrease strictly when we haven't reach an optimal solution and in case we have reached such a point, at most the energy remains constant through posterior iterations of the algorithm.

Typically one designs the iterative algorithm to make a certain energy or objective function to become a descent, for example, if we want to solve the program

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \Omega \end{aligned} \tag{A.1}$$

Then we can design an iterative algorithm $\mathcal{A}\colon \Omega \to \mathcal{P}(\Omega)$ such that the sequence $\{\mathbf{x}_n\}_{0,1,\ \cdots n, \cdots}$ satisfies $f(\mathbf{x}_{n+1}) < f(\mathbf{x}_n)$ if $\mathbf{x}_n$ is not an optimal solution and $\mathbf{x}_{n+1} = \mathbf{x}_n$ if $\mathbf{x}_n$ is an optimal solution. This way, the *solution set* Γ is the set of optimal solutions (local minima) and the *descent function* is f.

For the main result it is needed to find some kind of continuity condition for $\mathcal{A}$. In order for that we will define the graph of a point-to-set function Φ, inspired by the following proposition:

Proposition A.2.1. *Given two metric spaces X and Y and a function $f\colon X \to Y$, f is continuous if and only if $Gr(f) \subset X \times Y$ is closed in $X \times Y$*

(the graph of f is defined as $Gr(f) = \{(x, y) \subset X \times Y \mid y = f(x)\}$)
Therefore whenever $x_k \to x$ and $f(x_k) = y_k \to y$, $(x, y) = (x, f(x))$.
Now we define the graph of a point-to-set function Φ:

Definition A.2.4. *Given two metric spaces X and Y and a set-valued function $\Phi\colon X \to \mathcal{P}(Y)$, its graph is the set*

$$Gr(\Phi) = \{(x, y) \in X \times Y \mid y \in \Phi(x)\}$$

The graph is simply the collection $Gr(\Phi) = \cup\{x\} \times \Phi(x)$.
We are in position to define a notion of continuity of point-to-set mappings, inspired by the previous proposition:

Definition A.2.5. *A point-to-set mapping is closed at x_0 if whenever two sequences $x_k \to x_0$ and $y_k \to y_0$ such that $y_k \in \Phi(x_k)\ \forall\ k$ implies that $y_0 \in \Phi(x_0)$.*
If Φ is closed for every $x \in S$, we say it is closed on $S \subset X$.

This notion will be crucial for the main result of this section. Also, as many cluster algorithms are the iterations of two consecutive steps, we need next to analyze under what conditions the composition of two *iterative algorithms* are still continuous.

Definition A.2.6. *Let $\mathcal{A}\colon X \to \mathcal{P}(Y)$ and $\mathcal{B}\colon Y \to \mathcal{P}(Z)$ two point-to-set mapping. The composite map $\mathcal{C} = \mathcal{B} \circ \mathcal{A}\colon X \to \mathcal{P}(Z)$ is defined by*

$$\mathcal{C}(x) = \bigcup_{y \in \mathcal{A}(x)} \mathcal{B}(y)$$

Proposition A.2.2. *Let $\mathcal{A}\colon X \to \mathcal{P}(Y)$ and $\mathcal{B}\colon Y \to \mathcal{P}(Z)$ such that*

i. $\mathcal{A}$ is closed at x_0;

ii. $\mathcal{B}$ is closed on $\mathcal{A}(x_0)$;

iii. For every $x_k \to x_0$ and $y_k \in \mathcal{A}(x_k)$, there is a convergent subsequence $y_{k_l} \to y$.
Then $\mathcal{C} = \mathcal{B} \circ \mathcal{A}$ is closed at x_0

Proof. Let $x_k \to x_0$ and $z_k \to z$ such that $z_k \in \mathcal{C}(x_k)$. By the definition of $\mathcal{C}$, for each k one can pick a $y_k \in \mathcal{A}(x_k)$ such that $z_k \in \mathcal{B}(y_k)$.
By *iii.*, there is a convergent subsequence $y_{k_l} \to y$. As $x_{k_l} \to x_0$ and $y_{k_l} \in \mathcal{A}(x_{k_l})$, by *i.*, it implies that $y \in \mathcal{A}(x_0)$.
By *ii.*, $\mathcal{B}$ is closed at y, hence the limits $y_{k_l} \to y$ and $z_{k_l} \to z$ with $z_{k_l} \in \mathcal{B}(y_{k_l})$ imply that $z \in \mathcal{B}(y) \subset \mathcal{C}(x_0)$ □

Corollary A.2.0.1. *If $\mathcal{A}$ is closed at x_0 and $\mathcal{B}$ is closed on $\mathcal{A}(x_0)$, then, if Y is compact, $\mathcal{B} \circ \mathcal{A}$.*

Now we finally state the main result of this section:

Theorem A.2.1 (Zangwill Global Convergence Theorem)**.** *Let $\mathcal{A}\colon X \to \mathcal{P}(X)$ be an iterative algorithm together with x_0 and let $\{x_k\}_{k=0,1,2,\cdots}$ be a sequence satisfying*

$$x_{k+1} \in \mathcal{A}(x_k)$$

Also let a solution set $\Gamma \subset X$ be given and suppose that

i. $\{x_k\}_{k=0,1,2,\cdots} \subset S$ for $S \subset X$ compact;

ii. There exists a continuous function E which is also a $descent function$ for Γ;

iii. $\mathcal{A}$ is closed on X/Γ
Then the limit of any convergent subsequence of $\{x_k\}_{0,1,2,\cdots}$ lies in the solution set Γ.

Proof. Suppose there is a convergent subsequence of $\{x_k\}_{0,1,2,\ldots}$, such that $x_{k_l} \to x$ for some x, a limit point of $\{x_k\}$. By continuity of E, $E(x_{k_l}) \to E(x)$. We extend this result to show that it is valid for the whole sequence.
By *ii.*, E is a *descent function* thus satisfying $E(x_{k+1}) \leq E(x_k)$. So we have

$$E(x) = \lim_{l \to \infty} E(x_{k_l}) \leq E(x_{k_l})$$

Given $\epsilon > 0$, there exists a l_0 such that, for $l \geq l_0$,

$$|E(x_{k_l}) - E(x)| = E(x_{k_l}) - E(x) < \epsilon$$

For every $k > k_{l_0}$, $E(x_k) \leq E(x_{k_l})$, therefore,

$$E(x_k) - E(x) \leq E(x_{k_l}) - E(x) < \epsilon$$

Now, given such a k, we can select an index l_1 with $k_{l_1} > k$ and verify that

$$0 \leq E(x_{k_{l_1}}) - E(x) \leq E(x_k) - E(x) < \epsilon$$

Hence we proved that for every $\epsilon > 0$, there exists a l_0 such that if $k > k_{l_0}$, $|E(x_k) - E(x)| = E(x_k) - E(x) < \epsilon$ which is the same as to say

$$\lim_{k \to \infty} E(x_k) = E(x)$$

Now we affirm that $x \in \Gamma$. For suppose not. The sequence $\{x_{k_l+1}\}_{l=0,1,2,\cdots}$ has the property $x_{k_l+1} \in \mathcal{A}(x_{k_l})$ and is contained in the compact set S, hence there is a subsequence $\{x_{k_{l_m}+1}\} \subset \{x_{k_l+1}\}_{l=0,1,2,\cdots}$ converging to a $\bar{x} \in S$. this sequence is essentially different from $\{x_{k_l}\}$ and they do not necessarily converge to the same value, *i.e.*, not necessarily $x = \bar{x}$. Define the sequences

$$\{y_m\} = \{x_{k_{l_m}}\}_{m=0,1,2,\cdots}$$
$$\{z_m\} = \{x_{k_{l_m}+1}\}_{m=0,1,2,\cdots}$$

We have that $y_m \to x$ and $z_m \to \bar{x}$ with $z_m \in \mathcal{A}(y_m)$. $\mathcal{A}$ is closed at X/Γ and by hypothesis $x \in X/\Gamma$, therefore by continuity, $\bar{x} \in \mathcal{A}(x)$ and this implies, by the *descent function* property that $E(\bar{x}) < E(x)$.

On the other side, as $\{E(z_m)\} \subset \{E(x_k)\}$, which converges to $E(x)$, and E is continuous,

$$E(\bar{x}) = E(\lim_{m \to \infty} z_m) = \lim_{m \to \infty} E(z_m) = \lim_{k \to \infty} E(x_k) = E(x)$$

a contradiction, we conclude that x has to belong to the *solution set* Γ.

□

Zangwill's theorem is very general, so it really doesn't make it easier to prove the convergence of general algorithms, once it can be hard to check if conditions *i.-iii.* are met. The beauty and elegance of this result is to cast abstract algorithm methods represented by $\mathcal{A}$ in a unified systematic analysis.

Appendix B

2

B.1 Jensen Inequality

Consider a function $f\colon \mathbb{R} \to \mathbb{R}$. f is said to be convex if

$$f(a+t(b-a)) \leq f(a)+t(f(b)-f(a)) \ , \quad \forall t \in (0,1) \ , \quad a,b \in \mathbb{R} \tag{B.1}$$

And we say it is stricly convex if the inequality is strict
We want to prove that if $f''(x) \geq 0 \ \forall \ x$, then f is convex. First we prove the

Lemma B.1.1. *Given a function $f\colon [0,1] \to \mathbb{R}$, if $f''(x) \geq 0 \ \forall \ x$, such that $f(0) = f(1) = 0$ then f is convex.*

Proof. By the hypothesis on f'', we have that $f' = f'(x)$ is a monotone increasing function.
For the case in which $f \equiv constant$, it is trivially convex and clearly, $f \equiv 0$.
For the case $f \not\equiv constant$, we first we show that $f'(0) < 0$ and $f'(1) > 0$. Now, if $f'(0) \geq 0$, then, $0 \leq f'(0) \leq f'(x) \ \forall \ x$, thus, $f = f(x)$ is an increasing sequence. If $f' \equiv 0$, f becomes constant and equal to 0, a contradiction. If $f'(1) \leq 0$, then, $f'(x) \leq f'(1) \ \forall \ x$, thus, $f = f(x)$ is an increasing sequence, thus this function. If $f' \equiv 0$, f becomes constant and equal to 0, which is a degenerate case for us □

Theorem B.1.2 (Danskin)**.** *Let $(x,u) \mapsto f(x,u)$ be the function where $x \in S$ and $u \in \mathbf{R}^n$. Suppose S is a compact topological space, and that both f and $\partial f / \partial u^i$ are continuous. Define*

$$F(u) = \min_{x \in S}$$

A(u) = { x — x minimizes f(x,u) over S }

Also define the one-sided directional derivative of F in at u in the direction d as

$$DF(u;d) = \lim_{\alpha \to 0^+} \frac{F(u+\alpha d) - F(u)}{\alpha}$$

With these conditions, the one-sided directional derivative exists and is given by

$$DF(u;s) = \min_{x \in A(u)} \sum_{i=1}^{m} s_i \frac{\partial f}{\partial u_i}(x,u)$$

Proof. Let u be an arbitrary point of $\mathbb{R}^n$, and let $\{\alpha_k\}_{k\in\mathbb{N}}$ be a positive sequence tending to zero. Define

$$u_k = u + \alpha_k s$$

And consider a $x_k \in A(u_k)$ and $x \in A(u)$. We calculate

$$\frac{F(u_k) - F(u)}{\alpha_k} = \frac{f(x_k, u_k) - f(x,u)}{\alpha_k} = \frac{f(x_k, u_k) - f(x,u_k)}{\alpha_k} + \frac{f(x, u_k) - f(x,u)}{\alpha_k}$$

As $x_k \in A(u_k)$, $f(x_k, u_k) \leq f(x, u_k)$, hence $\frac{f(x_k,u_k)-f(x,u_k)}{\alpha_k} \leq 0$, so that

$$\frac{F(u_k) - F(u)}{\alpha_k} \leq \frac{f(x, u_k) - f(x,u)}{\alpha_k} \quad , \quad \forall\, x \in A(u)$$

By the mean value theorem,

$$\begin{aligned} \frac{f(x, u_k) - f(x,u)}{\alpha_k} &= \sum_{i=1}^{m} \frac{(u_k)_i - u_i}{\alpha_k} \frac{\partial f}{\partial u_i}(x, u + \theta_k(u_k - u)) \ , \quad 0 \leq \theta_k \leq 1 \\ \therefore \quad \frac{f(x, u_k) - f(x,u)}{\alpha_k} &= \sum_{i=1}^{m} d_k \frac{\partial f}{\partial u_i}(x, u + \theta_k(u_k - u)) \end{aligned}$$

And combining this with the previous inequality,

$$\frac{F(u_k) - F(u)}{\alpha_k} \leq \sum_{i=1}^{m} d_k \frac{\partial f}{\partial u_i}(x, u + \theta_k(u_k - u))$$

When $k \to \infty$, $\alpha_k \to 0$ so that $\theta_k(u_k - u) = \theta_k \alpha_k d \to 0$, which gives us

$$\lim_{k\to\infty} \sum_{i=1}^{m} d_k \frac{\partial f}{\partial u_i}(x, u + \theta_k(u_k - u)) = \sum_{i=1}^{m} d_k \frac{\partial f}{\partial u_i}(x,u)$$

Now, given a convergent subsequence $\{u_{k_l}\} \subset \{u_k\}$, in the limit $l \to \infty$ because of the above inequality, we must have that

$$\lim_{l\to\infty} \frac{F(u_k) - F(u)}{\alpha_k} \leq \sum_{i=1}^{m} d_k \frac{\partial f}{\partial u_i}(x,u) \ \ \forall\, x \in S$$

Choosing the special subsequence converging to the lim sup, we thus arrive at the result:

$$\limsup \frac{F(u_{k_l}) - F(u)}{\alpha_{k_l}} \leq \sum_{i=1}^{m} d_k \frac{\partial f}{\partial u_i}(x,u) \ \ \forall\, x \in S$$

Now we must construct the reverse inequality. Let's get any subsequence $\{u_{k_l}\}$ converging to $\liminf\,(F(u_{k_l}) - F(u))/\alpha_{k_l}$.

□

More can be seen at [18] [ch. 8, pg. 420] .

Appendix C

4

C.1 A short introduction to Optimization

In what follows we briefly introduce notions of optimization, very much in the style of the excellent reference [24].

C.2 Unconstrained Optimization

The most basic optimization program is to find the *minimum* of a function $f\colon \mathbb{R}^d \to \mathbb{R}$, *i.e.*, a point $x_{min} \in \mathbb{R}^d$ such that

$$f(\mathbf{x}_{min}) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbb{R}^d \tag{C.1}$$

if it exists.
We denote this program by

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \mathbb{R}^d \end{aligned} \tag{C.2}$$

In order to solve this problem, it is needed more knowledge over the structure of f. Normally it is assumed that it is at least twice differentiable. In practice, this assumption applies well to a bunch of important problems, but even in these cases, the calculation of the derivatives might be not straightforward.
Working with this assumption, if the minimum exists, we can infer from condition C.1 that for every unitary vector $\mathbf{u} \in \mathbb{R}^d$, $f(\mathbf{x_{min}}) \leq f(\mathbf{x_{min}} + t\mathbf{u})$. Therefore, if we define the function $h(t) = f(\mathbf{x_{min}} + t\mathbf{u})$, we get

$$0 = h'(0) = \nabla f(\mathbf{x_{min}}) \cdot \mathbf{u} \quad \forall\ \mathbf{u}\ \in \mathrm{\mathbb{R}^d}$$

Hence, $\nabla f(\mathbf{x_{min}}) = 0$. However, this is only a necessary condition for the minimum. Other points may satisfy this condition, as in the case of a point with the following property

Definition C.2.1. $\bar{\mathbf{x}}$ *is a local minimum for the function* f *if there is an open neighborhood* $V \ni \bar{\mathbf{x}}$ *such that* $f(\bar{\mathbf{x}}) \leq f(\mathbf{x})$, $\forall\ x \ \in V$

For such a point, if we get an $\epsilon > 0$ such that the open ball $B(\mathbf{x_{min}}, \epsilon) \subset V$, then for every $\mathbf{u} \in \mathbb{R}^{\mathrm{d}}$ unitary, $f(\mathbf{x_{min}}) \leq f(\mathbf{x_{min}} + t\mathbf{u})$, for all $t \in (-\epsilon, \epsilon)$, and therefore, the conclusion **??** remains valid.
We found a sufficient condition, and if there exists a minimum, which we now call *global minimum* in opposition to a *local minimum*, it surely is one among the points which satisfies $\nabla f(\bar{\mathbf{x}}) = 0$. There is another situation in which the derivative is zero although the point is not a (local) minimum: a saddle point. To characterise it we need the following

Definition C.2.2. *A unitary vector* $\mathbf{u}$ *is a descent direction for a point* $\mathbf{x} \in \mathbb{R}^{\mathrm{d}}$ *if there exists a* $\delta > 0$ *such that* $f(\mathbf{x} + t\mathbf{u}) < f(\mathbf{x})$*, for each* $t \in (0, \delta)$.

Notice that t lies on the open interval $(0, \delta)$, not $(-\delta, \delta)$.
Let us study the behavior of f in a neighborhood of some $\mathbf{x}$ with zero derivative by Taylor expanding to the next order, with second derivatives:

$$\begin{aligned} f(\mathbf{x} + t\mathbf{u}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T (t\mathbf{u}) + \frac{1}{2}\langle t\mathbf{u}, \mathbf{H}(\mathbf{x})(t\mathbf{u})\rangle + \|t\mathbf{u}\|^2 \rho_{\mathbf{x}}(\|t\mathbf{u}\|) \\ &= f(\mathbf{x}) + \frac{t^2}{2}\langle \mathbf{u}, \mathbf{H}(\mathbf{x})\mathbf{u}\rangle + t^2 \rho_{\mathbf{x}}(t) \end{aligned} \tag{C.3}$$

Where $\mathbf{H}(\mathbf{x}) = \left[\ \frac{\partial^2 f}{\partial x^i \partial x^j}(\mathbf{x})\ \right]$ is the Hessian matrix of f and $\rho_x(\|\mathbf{v}\|)$ tends to zero as $\|\mathbf{v}\| \to 0$. Now, the behavior of f in a vicinity of $\mathbf{x}$ depends completely on $\mathbf{H}(\mathbf{x})$. Let us diagonalize it and obtain its eigenvalues:

$$\mathbf{H}(\mathbf{x}) = \mathbf{P}\ \mathbf{diag}(\lambda_1, \lambda_2 \cdots \lambda_d)\mathbf{P}^T$$

Suppose that one of these eigenvalues is negative, say, the λ_i corresponding to the i^{th} normalized eigenvector $\mathbf{u}_i$. Plugging it in C.3 we have

$$\begin{aligned} f(\mathbf{x} + t\mathbf{u}_i) =& f(\mathbf{x}) + \frac{t^2}{2}\lambda_i \langle \mathbf{u}_i, \mathbf{u}_i\rangle + t^2 \rho_{\mathbf{x}}(t) \\ \Rightarrow& f(\mathbf{x} + t\mathbf{u}_i) - f(\mathbf{x}) = \frac{t^2}{2}(\lambda_i + 2\rho_{\mathbf{x}}(t)) \end{aligned}$$

As $\lim_{t\to 0} \rho_{\mathbf{x}}(t) = 0$, and $t^2 > 0$, one can choose an $\epsilon > 0$ such that $t^2(\lambda_i + 2\rho_{\mathbf{x}}(t))$ for each $t \in (-\epsilon, \epsilon)$ (for instance, find an $\epsilon > 0$ such that $|\rho_{\mathbf{x}}(t)| < \frac{|\lambda_i|}{4}$, $\forall\ |t| < \epsilon$), thus,

$$f(\mathbf{x} + t\mathbf{u}_i) < f(\mathbf{x})\ \forall\ t\ \in (-\epsilon, \epsilon)$$

We conclude that, even though $\nabla f(\mathbf{x}) = 0$, depending on the existence of negative eigenvalues of $\mathbf{H}(\mathbf{x})$, it could be possible to find a descent direction for $\mathbf{x}$. Let's state two theorems below, one for a necessary condition, another for a sufficient condition for local minima:

Theorem C.2.1. *Suppose $f\colon \mathbb{R}^{\mathrm{d}} \to \mathbb{R}$ is twice differentiable. If $\mathbf{x}$ is a local minimum, then, $\nabla f(\mathbf{x}) = 0$ and $\mathbf{H}(\mathbf{x})$ is positive semidefinite.*

Remark. *A matrix $\mathbf{H}$ is semidefinite if $\langle \mathbf{u}, \mathbf{H}\mathbf{u}\rangle \geq 0 \ \forall \ \mathbf{u} \in \mathbb{R}^{\mathrm{d}}$. We say it is definite if this condition can be made strict.*

Proof. It was already proved that $\nabla f(\mathbf{x}) = 0$, now we consider the Taylor expansion C.3 for an arbitrary unitary direction $\mathbf{u}$, and divide everything by t^2, giving us

$$\frac{f(\mathbf{x}+t\mathbf{u}) - f(\mathbf{x})}{t^2} = \frac{1}{2}\langle \mathbf{u}, \mathbf{H}(\mathbf{x})\mathbf{u}\rangle + \rho_{\mathbf{x}}(t)$$

By the local minimum condition, there is an $\epsilon > 0$ such that for every $|t| < \epsilon$, the left side of the equation above is positive. Therefore, in the limit $t \to 0$, we have

$$\frac{1}{2}\langle \mathbf{u}, \mathbf{H}(\mathbf{x})\mathbf{u}\rangle = \lim_{t\to 0}\left(\frac{1}{2}\langle \mathbf{u}, \mathbf{H}(\mathbf{x})\mathbf{u}\rangle + \rho_{\mathbf{x}}(t)\right) =$$
$$\lim_{t\to 0}\frac{f(\mathbf{x}+t\mathbf{u}) - f(\mathbf{x})}{t^2} \geq 0 \ , \quad \forall \ \mathbf{u} \in \mathbb{R}^{\mathrm{d}} \ , \quad \text{unitary}$$

For a general $\mathbf{v} \neq 0$, denoting the unitary direction of $\mathbf{v}$ by $\hat{\mathbf{v}} = \mathbf{v}/\|\mathbf{v}\|$, we conclude

$$\langle \mathbf{v}, \mathbf{H}(\mathbf{x})\mathbf{v}\rangle = \langle \|\mathbf{v}\|\hat{\mathbf{v}}, \mathbf{H}(\mathbf{x})(\|\mathbf{v}\|\hat{\mathbf{v}})\rangle = \|\mathbf{v}\|^2\langle \hat{\mathbf{v}}, \mathbf{H}(\mathbf{x})\hat{\mathbf{v}}\rangle > 0$$

□

Theorem C.2.2. *Suppose $f\colon \mathbb{R}^{\mathrm{d}} \to \mathbb{R}$ is twice differentiable. If $\nabla f(\mathbf{x}) = 0$ and $\mathbf{H}(\mathbf{x})$ is positive definite, then $\mathbf{x}$ is a local strict minimum.*

Proof. Expanding f in Taylor series, we get

$$f(\mathbf{y}) = f(\mathbf{x}) + \frac{1}{2}\langle(\mathbf{y}-\mathbf{x}), \mathbf{H}(\mathbf{x})(\mathbf{y}-\mathbf{x})\rangle + \|\mathbf{y}-\mathbf{x}\|^2\rho_{\mathbf{x}}(\|\mathbf{y}-\mathbf{x}\|) \tag{C.4}$$

The proof goes by contradiction: suppose that $\mathbf{x}$ is not a local minimum. Then, for every $\epsilon > 0$, we can find a $\mathbf{x}'$ such that $f(\mathbf{x}') \leq f(\mathbf{x})$ and $\|\mathbf{x}' - \mathbf{x}\| < \epsilon$. So, we can construct a sequence $\{\mathbf{x}_k\}_{k=1,2,\cdots}$ converging to $\mathbf{x}$ with $f(\mathbf{x}_k) \leq f(\mathbf{x})$ and $\mathbf{x}_k \neq \mathbf{x}_{k'}$, for every k and k'.
Define a new sequence by introducing

$$\mathbf{d}_k = \frac{\mathbf{x}_k - \mathbf{x}}{\|\mathbf{x}_k - \mathbf{x}\|}$$

As $\|\mathbf{d}_k\| = 1$, $\forall \ k$, by compactness of the sphere, there exists a subsequence $\{\mathbf{d}_{k_l}\} \subset \{\mathbf{d}_k\}$ and a $\mathbf{d}$ such that $\lim\limits_{l\to\infty} \mathbf{d}_{k_l} = \mathbf{d}$.
Now, substitute in equation C.4 the values $\mathbf{x}_{k_l} - \mathbf{x} = \|\mathbf{x}_{k_l} - \mathbf{x}\|\mathbf{d}_{k_l}$ and divide everything by $\|\mathbf{x}_{k_l} - \mathbf{x}\|$ forming

$$\frac{f(\mathbf{x}_{k_l}) - f(\mathbf{x})}{\|\mathbf{x}_{k_l} - \mathbf{x}\|^2} = \frac{1}{2}\langle \mathbf{d}_{k_l}, \mathbf{H}(\mathbf{x})\mathbf{d}_{k_l}\rangle + \rho_{\mathbf{x}}(\|\mathbf{x}_{k_l} - \mathbf{x}\|)$$

As $\mathbf{x}_{k_l} \to \mathbf{x}$ and $\mathbf{d}k_l \to \mathbf{d}$ when $l \to \infty$, $\frac{f(\mathbf{x}_{k_l}) - f(\mathbf{x})}{\|\mathbf{x}_{k_l} - \mathbf{x}\|^2} \geq 0$, by construction, then

$$\begin{aligned} 0 \geq \lim_{l\to\infty} \frac{f(\mathbf{x}_{k_l}) - f(\mathbf{x})}{\|\mathbf{x}_{k_l} - \mathbf{x}\|^2} &= \\ = \lim_{l\to\infty} \left(\frac{1}{2}\langle \mathbf{d}k_l, \mathbf{H}(\mathbf{x})\mathbf{d}k_l\rangle + \rho_{\mathbf{x}}(\|\mathbf{x}_{k_l} - \mathbf{x}\|) \right) &= \frac{1}{2}\langle \mathbf{d}, \mathbf{H}(\mathbf{x})\mathbf{d}\rangle \end{aligned}$$

Thus, reaching a contradiction, once by hypothesis $\langle \mathbf{d}, \mathbf{H}(\mathbf{x})\mathbf{d}\rangle > 0$. Therefore, the initial assertion is true.

□

Basically, we characterized completely a *local minimum* point by the properties f should satisfy at it. Whether a local minimum is also a *global minimum* or not, will depend on the global properties of the function. For example, if it is *convex*, there will be only one local minimum which is clearly global. More details of special cases can be found in reference [24].
Next, we discuss a more complex case where the optimization program is done over a restricted set of points.

C.3 A special case of Constrained Optimization

The previous result was easily achieved because the set of points is the whole $\mathbb{R}^{\mathrm{d}}$. The most general minimization program can be stated restricting the set of points to a subset of $\mathbb{R}^{\mathrm{d}}$:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \Omega \end{aligned} \tag{C.5}$$

In general, this is a too broad of a program. So, we restrict ourselves to the special case

$$\Omega = \{\mathbf{x} \in \mathbb{R}^{\mathrm{d}} \ \mid \ g(\mathbf{x}) = \alpha\} \equiv g^{-1}(\alpha)$$

Where $g \colon : U \to \mathbf{R}$ is some function defined over an open $U \subset \mathbf{R}$ and $\alpha \in \mathbf{R}$.
This subsection is pretty much in the spirit of the classical reference [28].
In order to solve it, we are going to transform this problem, at least locally, in an unconstrained problem. Suppose $\bar{\mathbf{x}}$ is an optimal solution for C.5, and that $\nabla g(\bar{\mathbf{x}}) \neq 0$. Without loss of generality, suppose that $\frac{\partial g}{\partial x^n}(\bar{\mathbf{x}}) \neq 0$, and let us decompose $\bar{\mathbf{x}}$ in the form $\bar{\mathbf{x}} = (x_{min}, y_{min})$ with $x_{min} \in \mathbf{R}^{d-1}$ and $y_{min} \in \mathbf{R}$. Then by the Implicit Function Theorem, there exists a function $\xi \colon V \to \mathbf{R}$ where V is an open neighborhood of x_{min} in $\mathbf{R}^{d-1}$, and there exists an open set $W \subset \mathbb{R}^{\mathrm{d}}$ such that

$$\mathbf{x} \in g^{-1}(\alpha) \cap W \Leftrightarrow \mathbf{x} = (x, \xi(x))$$

So, at least locally in V, we can simplify our problem to the following program:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & h(x) = f(x, \xi(x)) \\ \text{subject to} \quad & x \in V \end{aligned}$$

The legitimity of this transformation depends on the existence $\bar{\mathbf{x}}$ and knowledge of V and ξ. This program is almost the same as the unconstrained program C.2 but for a smaller open region. Therefore, the solution is expected to be the same as with the whole $\mathbb{R}^{\mathrm{d}}$. By studying only the critical points x_c for which $\nabla h(x_c) = 0$, we discover a relation between $\nabla f(\bar{\mathbf{x}})$ and $\nabla g(\bar{\mathbf{x}})$:

$$0 = \frac{\partial h}{\partial x^i}(x_c) = \frac{\partial f}{\partial x^i}(x_c, y_c) + \frac{\partial f}{\partial y}(x_c)\frac{\partial \xi}{\partial x^i}(x_c, y_c)$$

On the other side, by the definition of ξ,

$$\begin{aligned} g(x, \xi(x)) = \alpha \qquad \forall \;\; x \in V \\ \Rightarrow \frac{\partial g}{\partial x^i}(x) + \frac{\partial g}{\partial y}(x)\frac{\partial \xi}{\partial x^i}(x) = 0 \\ x_c \in V \Rightarrow \frac{\partial \xi}{\partial x^i}(x_c) = -\frac{\frac{\partial g}{\partial x^i}(x_c)}{\frac{\partial g}{\partial y}(x_c)} \end{aligned}$$

Therefore,

$$\frac{\partial f}{\partial x^i}(x_c, y_c) - (\frac{\partial f}{\partial y}(x_c)/\frac{\partial g}{\partial y}(x_c))\frac{\partial g}{\partial x^i}(x_c) = 0$$

Defining $\lambda = \frac{\partial f}{\partial y}(x_c)/\frac{\partial g}{\partial y}(x_c)$, we have the following relations:

$$\begin{cases} \frac{\partial f}{\partial y}(x_c) = \lambda\frac{\partial g}{\partial y}(x_c) \\ \\ \frac{\partial f}{\partial x^i}(x_c, y_c) = \lambda\frac{\partial g}{\partial x^i}(x_c) \end{cases}$$

Finally, we conclude that the following necessary condition relating f and g through their first derivatives holds:

$$\nabla f(\bar{\mathbf{x}}) = \lambda \nabla g(\bar{\mathbf{x}})$$

Geometrically, this means that the gradient of f at the minimum over the surface $\Omega = g^{-1}(\alpha)$ is normal to its tangent surface at $\bar{\mathbf{x}}$ (as Ω is a level surface of g, ∇g points in the normal direction out of it).

C.4 Optimization with equality constraints

We now extend the last result to the more complex program where $\mathbf{x}$ is restricted to m constraints:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) = \alpha_i \ , \quad i = 1, 2, \cdots m \end{aligned} \tag{C.6}$$

We do so by proving the

Theorem C.4.1. *Let the constraints $g_i \colon \mathbb{R}^{\mathrm{d}} \to \mathbb{R}$, $i = 0, 1, \cdots m$ ($m < d$), be continuously differentiable functions and let $\bar{\mathbf{x}}$ be a minimum of the problem C.6. If the derivatives $\nabla g_i(\bar{\mathbf{x}})$, $i = 1, 2, \cdots m$ are linearly independent, then there exists constants $\lambda_1, \lambda_2, \cdots \lambda_m$ such that*

$$\nabla f(\bar{\mathbf{x}}) = \lambda_1 \nabla g_1(\bar{\mathbf{x}}) + \lambda_2 \nabla g_2(\bar{\mathbf{x}}) + \cdots + \lambda_m \nabla g_m(\bar{\mathbf{x}})$$

Remark. *Defining the function $F \colon \mathbb{R}^{\mathrm{d}} \to \mathbb{R}^m$ by $F(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \cdots g_m(\mathbf{x}))$ and the vector $\alpha = (\alpha_1, \alpha_2, \cdots \alpha_m)$, the restriction of C.6 can be seen as the $(d - m)$-dimensional surface $\Omega = F^{-1}(\alpha)$.*
The linear independence of the set $\{\nabla g_1(\bar{\mathbf{x}}), \nabla g_2(\bar{\mathbf{x}}), \cdots \nabla g_m(\bar{\mathbf{x}})\}$ is equivalent to say that the rank of the jacobian matrix $DF(\bar{\mathbf{x}}) = \left[\ \frac{\partial g_i}{\partial x^j}(\bar{\mathbf{x}}) \ \right]$ is m.

Proof. By the remark, the rank of $\left[\ \frac{\partial g_i}{\partial x^j}(\bar{\mathbf{x}}) \ \right]$ is m, and by re-labelling the coordinates if necessary, we can assume that the sub-matrix $\left[\ \frac{\partial g_i}{\partial x^j}(\bar{\mathbf{x}}) \ \right]$, $1 \leq i, j \leq m$ is non-singular. By the implicit function theorem, there exists an open neighborhood W for $\bar{\mathbf{x}}$, an open set $V \subset \mathbb{R}^{d-m}$ and a function $\xi \colon V \to \mathbb{R}^m$ such that

$$g_i(\mathbf{x}) = \alpha_i \ , \ i = 1, 2, \cdots m \ \ \textit{and} \ \ \mathbf{x} \in W$$

If and only if

$$\begin{gathered} x^j = \xi^j(x^{m+1}, \cdots \ x^d) \ , \quad j = 1, 2, \cdots \ m \\ \textit{and} \ \ (x^{m+1}, \cdots \ x^d) \in V \\ \text{for } \mathbf{x} = (x^1, \cdots \ x^m; x^{m+1}, \cdots \ x^d) \end{gathered}$$

Define $u = (x^{m+1}, x^{m+2}, \cdots \ x^d)$ and $w = (x^1, x^2, \cdots \ x^m)$. So ξ satisfies

$$\begin{gathered} g_i(\xi(u), u) = g_i(\xi^1(u), \cdots \ \xi^m(u); u) = \alpha_i \\ \text{for } i = 1, 2, \cdots \ m \text{ and } \forall \ u \ \in V \end{gathered} \tag{C.7}$$

Then, the optimal decision can be written as $\bar{\mathbf{x}} = (\bar{w}, \bar{u}) = (\xi(\bar{u}), \bar{u})$ and is a solution of the transformed problem

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & h(u) = f(\xi(u), u) \\ \text{subject to} \quad & u \in V \end{aligned}$$

As ξ and f are differentiable (by hypothesis and the Implicit Function Theorem), then the optimal solution satisfies

$$0 = \frac{\partial h}{\partial u^i}(\bar{u}) = \frac{\partial f}{\partial u^i}(\bar{\mathbf{x}}) + \sum_{j=1}^{m} \frac{\partial f}{\partial w^j}(\bar{\mathbf{x}}) \frac{\partial \xi^j}{\partial u^i}(\bar{u}) \ \forall \ \ i = 1, 2, \cdots \ m$$

Differentiating expression C.6 we get

$$\frac{\partial g_j}{\partial u^i}(\xi(\bar{u}), \bar{u}) = \frac{\partial g_j}{\partial u^i}(\bar{\mathbf{x}}) + \sum_{k=1}^{m} \frac{\partial g_j}{\partial w^k}(\bar{\mathbf{x}}) \frac{\partial \xi^k}{\partial u^i}(\bar{u}) = \frac{\partial \alpha_j}{\partial u^i} = 0$$
$$\text{for } 1 \leq j \leq m \text{ and } m+1 \leq i \leq d$$

Adopting the matrix notations

$$D_u F(\bar{\mathbf{x}}) = \left[\begin{array}{c} \frac{\partial g_j}{\partial u^i}(\bar{\mathbf{x}}) \end{array} \right] \ , \quad D_w F(\bar{\mathbf{x}}) = \left[\begin{array}{c} \frac{\partial g_j}{\partial w^k}(\bar{\mathbf{x}}) \end{array} \right] \ , \quad D\xi(\bar{u}) = \left[\begin{array}{c} \frac{\partial \xi^j}{\partial u^i}(\bar{u}) \end{array} \right]$$

We restate the above equalities in a more compact form:

$$\nabla_u f(\bar{\mathbf{x}}) + D\xi(\bar{u}) \cdot \nabla_w f(\bar{\mathbf{x}}) = 0 \quad (1)$$
$$D_u F(\bar{\mathbf{x}}) + D\xi(\bar{u}) \cdot D_w F(\bar{\mathbf{x}}) = 0 \quad (2)$$

By the condition (2), and on the non-singularity hypothesis of $D_w F(\bar{\mathbf{x}})$, inverting it we arrive at an expression for the *a priori* unknown matrix $D\xi(\bar{u})$:

$$D\xi(\bar{u}) = -D_u F(\bar{\mathbf{x}}) \cdot D_w F(\bar{\mathbf{x}})^{-1}$$

Hence, by applying this on the previous result in condition (1),

$$\nabla_u f(\bar{\mathbf{x}}) - D_u F(\bar{\mathbf{x}}) \cdot D_w F(\bar{\mathbf{x}})^{-1} \cdot \nabla_w f(\bar{\mathbf{x}}) = 0$$
$$\Rightarrow \nabla_u f(\bar{\mathbf{x}})^T \equiv D_u F(\bar{\mathbf{x}})^T \cdot \lambda$$

Where we defined $\lambda^T = (\lambda_1, \lambda_2, \cdots \ \lambda_m) = (D_w F(\bar{\mathbf{x}})^{-1} \cdot \nabla_w f(\bar{\mathbf{x}}))^T$. Noticing that

$$D_u F(\bar{\mathbf{x}})^T = \left[\begin{array}{ccc} \nabla_u g_1(\bar{\mathbf{x}})^T & \cdots & \nabla_u g_m(\bar{\mathbf{x}})^T \end{array} \right]$$

We conclude the first part of our assertion:

$$\nabla_u f(\bar{\mathbf{x}}) = \lambda_1 \nabla_u g_1(\bar{\mathbf{x}}) + \cdots + \lambda_m \nabla_u g_m(\bar{\mathbf{x}})$$

The second part (for the derivatives in w) comes from the very definition of λ:

$$\lambda^T = (D_w F(\bar{\mathbf{x}})^{-1} \cdot \nabla_w f(\bar{\mathbf{x}}))^T$$
$$\Rightarrow \nabla_w f(\bar{\mathbf{x}})^T = (D_w F(\bar{\mathbf{x}})^T)^{-1} \cdot \lambda$$
$$\Leftrightarrow \nabla_w f(\bar{\mathbf{x}}) = \lambda_1 \nabla_w g_1(\bar{\mathbf{x}}) + \cdots + \lambda_m \nabla_w g_m(\bar{\mathbf{x}})$$

Which proves the assertion

$$\nabla f(\bar{\mathbf{x}}) = \lambda_1 \nabla g_1(\bar{\mathbf{x}}) + \cdots + \lambda_m \nabla g_m(\bar{\mathbf{x}})$$

□

Geometrically this result means that the derivative of f at $\bar{\mathbf{x}}$ should be perpendicular to the tangent planes of every surface level $g_i^{-1}(\alpha_i)$, meaning that the only decreasing directions could be pointing out of these surfaces.

C.5 Optimization with equality and inequality constraints

Finally we analyse the more general optimization program

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0 \ , \quad i = 1, 2, \cdots m \\ & h_j(\mathbf{x}) = 0 \ , \quad j = 1, 2, \cdots l \end{aligned} \tag{C.8}$$

We derive next a set of necessary conditions for an optimal solution for C.8, in which case we say it is a Karush-Kuhn-Tucker (KKT) point.

Of course, for a complete optimization analysis, one should look for sufficient conditions for a *minimium.* In our case, looking for KKT points will suffice, as the functions under consideration satisfies some convexity properties that single out the global minimum from the set of solutions for the KKT conditions.

Before proceeding, we'll make use of two important theorems: the *Hyperplane Separation Theorem* and the *Gordan's Theorem*, stated below.

Theorem C.5.1 (Hyperplane Separation)**.** *Let A and B ne two disjoint nonempty convex sets of $\mathbb{R}^{\mathrm{d}}$. Then there exists a nonzero vector $\mathbf{v}$ and a real number c such that*

$$\langle \mathbf{x}, \mathbf{v} \rangle \geq c \textit{ and } \langle \mathbf{y}, \mathbf{v} \rangle \leq c$$

for all $x \in A$ and $y \in B$.

Theorem C.5.2 (Gordan's)**.** *Exactly one of the following systems has a solution:*

$$\begin{aligned} &(1) \quad A^T y > 0 \textit{ for some } y \in \mathbb{R}^{\mathrm{d}} \\ &(2) \quad Ax = 0 \ , \quad x \geq 0 \textit{ for some non-zero } x \in \mathbb{R}^{\mathrm{d}} \end{aligned}$$

First we prove the

Lemma C.5.3. *Consider problem C.8, and a local optimal solution $\bar{\mathbf{x}}$ for it.*

$$\begin{aligned} I(\bar{\mathbf{x}}) &= \{ i \in \{1, 2, \cdots m\} \mid g_i(\bar{\mathbf{x}}) = 0 \} \\ F_0 &= \{ \mathbf{d} \in \mathbb{R}^{\mathrm{d}} \mid \langle \nabla f(\bar{\mathbf{x}}), \mathbf{d} \rangle < 0 \} \\ G_0 &= \{ \mathbf{d} \in \mathbb{R}^{\mathrm{d}} \mid \langle \nabla g_i(\bar{\mathbf{x}}), \mathbf{d} \rangle < 0 \ \textit{ for } i \in I(\bar{\mathbf{x}}) \} \\ H_0 &= \{ \mathbf{d} \in \mathbb{R}^{\mathrm{d}} \mid \langle \nabla h_j(\bar{\mathbf{x}}), \mathbf{d} \rangle = 0 \ \textit{ for } j = 1, 2, \cdots l \} \end{aligned}$$

If $\nabla h_j(\bar{\mathbf{x}}) \ \ j = 1, 2, \cdots l$ are a linear independent, then

$$F_0 \cap G_0 \cap H_0 = \varnothing$$

Proof. Because of the conditions $h_j(\bar{\mathbf{x}}) = 0$, this lemma becomes a bit hash to prove.
Suppose, by contradiction that there exists a vector $\mathbf{y} \in \mathbb{R}^{\mathrm{d}}$ such that $\langle \nabla f(\bar{\mathbf{x}}), \mathbf{y} \rangle < 0$; $\langle \nabla g_i(\bar{\mathbf{x}}), \mathbf{y} \rangle < 0$, for all $i \in I(\bar{\mathbf{x}})$; and $\langle \nabla h_j(\bar{\mathbf{x}}), \mathbf{y} \rangle = 0$ for all $j = 1, 2, \cdots l$.
For a given $\mathbf{x} \in \mathbb{R}^{\mathrm{d}}$ consider the jacobian $DH(\mathbf{x}) = \begin{bmatrix} \nabla h_1(\mathbf{x})^T & \cdots & \nabla h_l(\mathbf{x})^T \end{bmatrix}^T$ (the rows are the gradients of h_j) and the projection operator $P(\mathbf{x})\colon \mathbb{R}^{\mathrm{d}} \to N_{DH(\mathbf{x})}$ where $N_{DH(\mathbf{x})}$ is the null space for $DH(\mathbf{x})$.
Define a curve $\alpha\colon s \mapsto \alpha(s) \in \mathbb{R}^{\mathrm{d}}$ as the solution of the following differential equation with boundary condition:

$$\begin{cases} \frac{d\alpha}{ds}(s) = P(\alpha(s))\mathbf{y} \\ \\ \alpha(0) = \bar{\mathbf{x}} \end{cases}$$

This equation is well defined for sufficiently small s, because $P(\mathbf{x})$ can be shown to be continuous at $\bar{\mathbf{x}}$ (from the differentiability of the h_j's and the linear independence of the ∇h_j's).
The idea is to prove that for sufficiently small s, $\alpha(s)$ is a feasible point, and that $f(\alpha(s)) < f(\bar{\mathbf{x}})$, a contradiction.
By the chain rule, we get, for every $i \in I(\bar{\mathbf{x}})$:

$$\begin{aligned} \frac{\partial(g_i \circ \alpha)}{\partial s}(s) &= \nabla g_i(\alpha(s)) \cdot \alpha'(s) = \\ &= \nabla g_i(\alpha(s)) \cdot P(\alpha(s))\mathbf{y} \end{aligned}$$

Specially for $s = 0$, $P(\alpha(0))\mathbf{y} = P(\bar{\mathbf{x}})\mathbf{y} = \mathbf{y}$, because $\mathbf{y}$ lies in the null space of $DH(\bar{\mathbf{x}})$, hence $(\partial g_i \circ \alpha / \partial s)(0) = \nabla g_i(\bar{\mathbf{x}})\mathbf{y} < 0$, so that for sufficiently small s it implies that

$$g_i(\alpha(s)) < 0$$

And for $i \notin I(\bar{\mathbf{x}})$, as $g_i(\bar{\mathbf{x}}) < 0$, the same result follows by continuity of g_i and α.
Now for the h_j's we apply again the chain rule, for sufficiently small s:

$$\begin{aligned} \frac{\partial h_j \circ \alpha}{\partial s}(s) &= \nabla h_j(\alpha) \cdot \alpha'(s) = \\ &= \nabla h_j(\alpha) \cdot P(\alpha(s))\mathbf{y} = 0 \quad \text{(by definition of } P\text{)} \\ &\Rightarrow h_j \circ \alpha(s) = \text{ const. } = h_j \circ \alpha(0) = h_j(\bar{\mathbf{x}}) = 0 \end{aligned}$$

For each $j = 1, 2, \cdots l$, $h_j(\alpha(s)) = 0$, and together with the previous result for the g_i's, considering sufficiently small s, we see that every point $\alpha(s)$ is a feasible point to problem C.8.
Next we calculate

$$\begin{aligned} \frac{\partial f \circ \alpha}{\partial s}(s) &= \nabla f(\alpha(s)) \cdot \alpha'(s) = \\ &= \nabla f(\alpha(s)) \cdot P(\alpha(s))\mathbf{y} \end{aligned}$$

For $s = 0$, $P(\alpha(0))\mathbf{y} = P(\bar{\mathbf{x}})\mathbf{y} = \mathbf{y}$, then $(\partial f \circ \alpha / \partial s)(0) = \nabla f(\bar{\mathbf{x}})\mathbf{y} < 0$. Therefore, for sufficiently small s, we have

$$f(\alpha(s)) < f(\alpha(0)) = f(\bar{\mathbf{x}})$$

But by definition of an optimal point, we arrive further at the contradiction

$$f(\alpha(s)) < f(\bar{\mathbf{x}}) \leq f(\alpha(s))$$

Hence, our original assertion is proven

□

Now we can proceed and prove the

Theorem C.5.4 (Karush-Kuhn-Tucker Necessary Conditions)**.**
Let $\bar{\mathbf{x}}$ be a feasible solution for C.8. Suppose that f and g_i for $i \in I(\bar{\mathbf{x}})$ are differentiable at $\bar{\mathbf{x}}$; that for $i \notin I(\bar{\mathbf{x}})$ g_i is continuous; and that each h_j for $j = 1, 2, \cdots l$ is continuously differentiable at $\bar{\mathbf{x}}$. Further, suppose that $\nabla g_i(\bar{\mathbf{x}})$, $i \in I(\bar{\mathbf{x}})$ and $\nabla h_j(\bar{\mathbf{x}})$, $j = 1, 2, \cdots l$ are linearly independent vectors.
If $\bar{\mathbf{x}}$ is a local optimal solution of C.8 then there exists unique scalars u_i for $i \in I(\bar{\mathbf{x}})$ and v_j for $j = 1, 2, \cdots l$ such that

$$\nabla f(\bar{\mathbf{x}}) + \sum_{i \in I(\bar{\mathbf{x}})} u_i \nabla g_i(\bar{\mathbf{x}}) + \sum_{j=1}^{l} v_j \nabla h_j(\bar{\mathbf{x}}) = 0$$
$$u_i \geq 0$$

Proof. Define the matrices

$$\begin{aligned} A_1 &= \begin{bmatrix} \nabla f(\bar{\mathbf{x}})^T & \nabla g_{i_1}(\bar{\mathbf{x}})^T & \cdots & \nabla g_{i_k}(\bar{\mathbf{x}})^T \end{bmatrix}^T_{i_j \in I(\bar{\mathbf{x}})} \\ A_2 &= \begin{bmatrix} \nabla h_1(\bar{\mathbf{x}})^T & \cdots & \nabla h_m(\bar{\mathbf{x}})^T \end{bmatrix}^T \end{aligned}$$

(for each matrix, the gradients compose its rows)
By the previous lemma, the system

$$A_1\mathbf{d} < 0 \ , \quad A_2\mathbf{d} = 0$$

has no solution. Consider the two sets

$$\begin{aligned} S_1 &= \{(\mathbf{z}_1, \mathbf{z}_2) \mid \mathbf{z}_1 = A_1\mathbf{y} \text{ and } \mathbf{z}_2 = A_2\mathbf{y} \text{ for } \mathbf{y} \in \mathbb{R}^{\mathrm{d}}\} \\ S_2 &= \{(\mathbf{z}_1, \mathbf{z}_2) \mid \mathbf{z}_1 < 0 \text{ and } \mathbf{z}_2 = 0\} \end{aligned}$$

It is straightforward to check that they are convex, that $0 \in \bar{S_1} \cap \bar{S_2}$, and clearly $S_1 \cap S_2 = \varnothing$ (otherwise the system above would have solution). Hence, by the Hyperplane Separation Theorem, there exists a non-zero vector $(\mathbf{p}_1, \mathbf{p}_2)$ such that

$$\begin{aligned} \langle \mathbf{p}_1, A_1\mathbf{y}\rangle + \langle \mathbf{p}_2, A_2\mathbf{y}\rangle \geq \langle \mathbf{p}_1, \mathbf{z}_1\rangle + \langle \mathbf{p}_2, \mathbf{z}_2\rangle \\ \forall \ \mathbf{y} \in \mathbb{R}^{\mathrm{d}} \ , \quad (\mathbf{z}_1, \mathbf{z}_2) \in \bar{S_2} \end{aligned} \tag{C.9}$$

Choosing $\mathbf{y} = 0$ and remembering that $\mathbf{z}_2 = 0$, we have the inequality $0 \geq \langle \mathbf{p}_1, \mathbf{z}_1\rangle$. As $\mathbf{z}_1$ can be an arbitrarily large negative vector, it thus implies that $\mathbf{p}_1 \geq 0$. Making $\mathbf{z}_1 = 0$ and $\mathbf{z}_2 = 0$ in C.9 we can rewrite it as

$$\langle A_1^T\mathbf{p}_1 + A_2^T\mathbf{p}_2, \mathbf{y}\rangle \geq 0 \ , \quad \forall \ \mathbf{y}$$

Inserting $\mathbf{y} = -(A_1^T\mathbf{p}_1 + A_2^T\mathbf{p}_2)$, we arrive at $-\|A_1^T\mathbf{p}_1 + A_2^T\mathbf{p}_2\|^2 \geq 0$, thus

$$A_1^T\mathbf{p}_1 + A_2^T\mathbf{p}_2 = 0$$

Naming the components of the $\mathbf{p}$'s as

$$(\mathbf{p}_1, \mathbf{p}_2) \equiv (\hat{u}_0, \hat{u}_{i_1}, \cdots \hat{u}_{i_k}; \hat{v}_1, \hat{v}_2, \cdots \hat{v}_l) \quad , \quad \text{where } i_j \in I(\bar{\mathbf{x}})$$

the above result can be stated explicitly as

$$\hat{u}_0 \nabla f(\bar{\mathbf{x}}) + \sum_{i \in I(\bar{\mathbf{x}})} \hat{u}_i \nabla g_i(\bar{\mathbf{x}}) + \sum_{j=1}^{l} \hat{v}_j \nabla h_j(\bar{\mathbf{x}}) = 0 \tag{C.10}$$

Now, if $\hat{u}_0 = 0$, then, by the linear independence of $\{\nabla g_i(\bar{\mathbf{x}}); \nabla h_j(\bar{\mathbf{x}})\}_{i \in I(\bar{\mathbf{x}}), j=1,\cdots l}$, the values for $\hat{u}_i$ and $\hat{v}_j$ would be zero, thus making $(\mathbf{p}_1, \mathbf{p}_2) = (\mathbf{0}, \mathbf{0})$, a contradiction. So $\hat{u}_0 > 0$. Now, define $u_i = \hat{u}_i / \hat{u}_0$ and $v_j = \hat{v}_j / \hat{v}_0$. Of course, $u_i \geq 0$. Dividing C.10 by $\hat{u}_0$ we conclude

$$\begin{aligned} \nabla f(\bar{\mathbf{x}}) + \sum_{i \in I(\bar{\mathbf{x}})} u_i \nabla g_i(\bar{\mathbf{x}}) + \sum_{j=1}^{l} v_j \nabla h_j(\bar{\mathbf{x}}) &= 0 \\ u_i &\geq 0 \end{aligned}$$

□

The above necessary condition can be restated in a nicer form. If for each $i \notin I(\bar{\mathbf{x}})$, g_i is continuous at $\bar{\mathbf{x}}$, then defining for each of these indices $u_i := 0$, we see that these coefficients must satisfy the conditions

$$\begin{aligned} \nabla f(\bar{\mathbf{x}}) + \sum_{i=1}^{m} u_i \nabla g_i(\bar{\mathbf{x}}) + \sum_{j=1}^{l} v_j \nabla h_j(\bar{\mathbf{x}}) &= 0 \\ u_i g_i(\bar{\mathbf{x}}) = 0 \ , \ u_i &\geq 0 \\ \text{for } i = 1, 2, \cdots &m \end{aligned} \tag{C.11}$$

From now on, we will refer to C.11 as the Karush-Kuhn-Tukker (KKT) necessary conditions. Before designing any algorithm for an optimization program , we'll first look to its KKT points, *i.e.*, the points for which C.11 has a solution. In special cases like when f is convex, and when g_i and h_j satisfy similar conditions, it can be shown that if $\bar{\mathbf{x}}$ satisfy the KKT conditions, it should be a global optimum solution for the program.

Appendix D

6

D.1 A Measure Concentration conjecture

We develop with more depth the brief statement made at section 3.0.2.3, 'ORCLUS'. The ORCLUS algorithm is designed to capture the relevant dimensions at each cluster locality. In this appendix we will try to put on solid grounds the following idea: at high dimensions, the σ_i eigenvalues of distributions like $\rho(x|i) = \rho(x|\Sigma_i, \mu_i, \theta_i)$ tend to be sparse.
An important theorem of concentration of measure is the generalization of *Hoeffiding's inequality*:

Theorem D.1.1 (Lévi - concentration of Lipschitz functions). *Let $S^{d-1} \subset \mathbb{R}^d$ be the unit sphere, $f\colon S^{d-1} \to \mathbb{R}$ be a λ-Lipschitz function in the L^2 norm and μ the uniform distribution over the sphere. Then,*

$$\mu(\ [f \geq med(f) + \epsilon]\ ) \leq 4\mathrm{e}^{-\epsilon^2 d/2\lambda^2}$$

Where med(f) is the median of f ($\mu(\ [f \leq med(f)]\ ) = \frac{1}{2}\mu(S^{d-1})$).

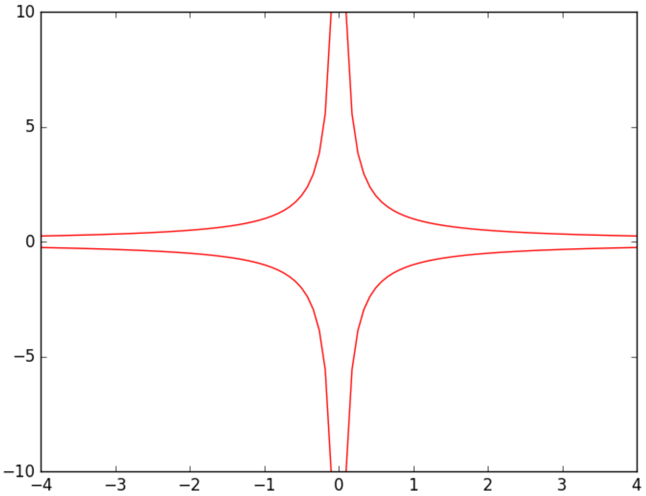
Now, let's work with the possibility of a similar result for our purposes. Here we deal with formal guesses of nice possibilities - so, we are not dealing with solid, rigorous mathematics, though these suggestions might shed some light on the necessity of an algorithm like PROCLUS.
Assume we are given a distribution with parameters Σ, μ and θ, the last one independent of the dimension d, and $\mu = E[\ X\ ]$, $\Sigma = E[\ (X-\mu)^T(X-\mu)\ ]$. Consider the set

$$S(r) = \{A \in \mathcal{M}_n(\mathbb{R}) \mid \|A\|_F = r\}$$

In the equivalence $\mathcal{M}_d(\mathbb{R}) \cong \mathbb{R}^{d^2}$, $\|\cdot\|_F$ corresponds to $\|\cdot\|_L^2$. Now, define the function

$$\begin{array}{rcl} f\colon S(r) & \to & \mathbb{R} \\ A & \mapsto & f(A) = \det(A) \end{array}$$

Let's admit that f satisfies the



Conjecture D.1.1. *Let μ be the uniform metric over $S(r)$ and $f(A) = \det(A)$. Then,*

$$\mu(\ [\ |f - med(f)| \geq \epsilon\ ]\ ) \leq O(\epsilon^m, d^n, r)$$

For some O of ϵ, d and r such that $O \overset{d\to\infty}{\longrightarrow} 0$.

The previous assertion may not be true, but we could wonder when and how something similar would be possible. But concentration of measure phenomena is so common and general, that it is not difficult to imagine that conjecture D.1.1 could be true. Assume this is the case. Then for large d, $\det(A) \approx const(r)$. On the other side, $det(A) = \lambda_1\lambda_2\cdots\lambda_d$, so we have

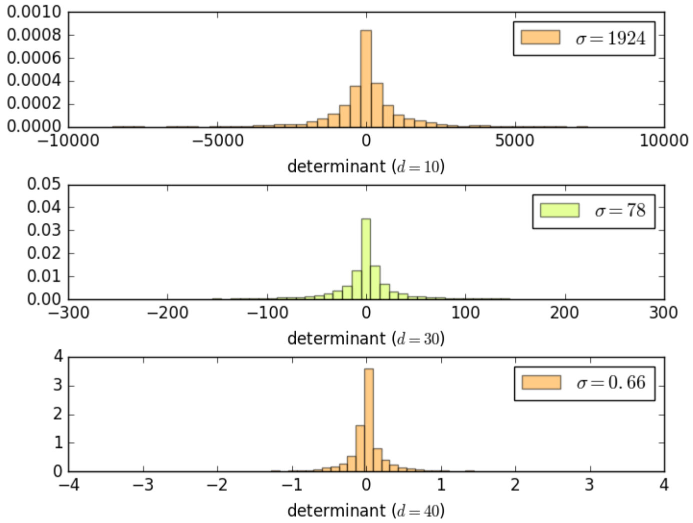
$$\lambda_1\lambda_2\cdots\lambda_d \approx const(r)$$

Define (at least locally for A) a function $h\colon A \to h(A) = (\lambda_1(A), \lambda_2(A), \cdots \lambda_d(A))$, where $\lambda_i(A)$ is the i-th eigenvalue for A (h needs to be defined exactly to do not create ambiguity with enumeration of eienvalues in the locality of A).

The region $\{\lambda_1\lambda_2\cdots\lambda_d = const(r)\}$ has the aspect of figure **??** (for a 2-d cut), which shows that in general the eigenvalues are sparse in the sense that $h(A)$ assumes values close to 0 and some very high eigenvalues. Conjecture D.1.1 says, in rough terms

$$\mu(\ h^{-1}(\ \{\lambda_1\lambda_2\cdots\lambda_d \in (const(r) - \delta, const(r) + \delta)\}\ )\ ) \approx O(\epsilon^m, d^n, r)$$

Empiric evidence is provided to show that indeed the determinant concentrates on the sphere.



D.2 A Pythonic ORCLUS implementation

```
class OrClus(object):
# self.c ; self.E ; self.S ; self.alpha ; self.delta
  def __init__(self,X,k,l,alpha=0.7):
    # X = n_samples x n_features numpy array,
    # X = [x1,x2,x3, ... ] , xi = [xi1,xi2, ... ]
    self.k = k
    self.l = l
    self.alpha=alpha
    self.dist = dist

  def OrClus(self,X=None,k=None,l=None,delta=10):

  ################## main ##########################

    if (X==None)&(k==None)&(l==None):
      # delta should be > 1! Ensuring that k0 > k.
      # initial k0 is delta times bigger than k
      self.k0 = int(np.ceil(delta*self.k))
      # initial l0 is equal to full dimension d
      self.l0 = len(self.X[0])
      # alpha
      a = self.alpha
      # calculates beta
      b = math.log(self.l0/self.l)*math.log(1/a)
      b/=math.log(self.k0/self.k)
      b = math.exp(-b)
      n = self.X.shape[0]
    # selects k0 random centroids among the data set X
    sample = random.sample(range(n),self.k0)

    # start with the selected centroids and identity matrices
    self.S = dict()
    self.E = dict()
    for i in range(len(sample)):
      self.S[i+1] = self.X[sample[i]]
      self.E[i+1] = np.eye(self.l0)

    # starts while loop
    while self.k0 > self.k:
      # skip further to assign method for comments
      self.Assign()

      #calculates the projected space using FindVectors method
      for i in range(self.k0):
        C = self.X[self.c==i+1]
        # checks if there are more than one vectors
        # (else we cannot calculate a covariance matrix)
        if C.shape[0]==1:
          self.E[i+1] = self.E[i+1]
        elif C.shape[0]>1:
          self.E[i+1] = self.FindVectors(C,self.l0)
        else:
          print("Empty Cluster!")

      # updates k (k(t)) to k_new (k(t+1))
      k_new = max(self.k,int(np.floor(a*self.k0)))
      # updates l (l(t)) to l_new (l(t+1))
      l_new = max(self.l,int(np.floor(b*self.l0)))
      # merges the clusters which minimizes the most the
      # ’projected energy’ cost function, until k and l
      # reduce to k_new and l_new.
      self.Merge(k_new,l_new)
      self.l0 = l_new

    # finally the last assignment
    self.Assign()

############# end main ##########################
############# procedures ########################

def Merge(self,k_new,l_new):
  n = len(self.c)
  while self.k0 > k_new:
    # starts merging the first two clusters
    # to initialize the loop minimization of
    # the ’projected energy’ r
```

```
      C = self.X[(self.c==1)|(self.c==2)]
      # recalculates the projected space
      E = self.FindVectors(C,l_new)
      # calclates the associated 'projected energy' cost function
      C -= C.mean(axis=0)
      s = np.linalg.norm(E.dot(C.T),axis=0).mean()
      t = (1,2)
      # starts merging iteration process. This piece of code
      # is highly paralellizable, as each such iteration in
      # the FOR loop can be calculated separately (only the
      # minimization of r should be performed  quickly thereafter)
      for i,j in itertools.combinations(range(self.k0),2):
        C = self.X[(self.c==i+1)|(self.c==j+1)]
        E = self.FindVectors(C,l_new)
        C -= C.mean(axis=0)
        # projected energy
        r = np.linalg.norm(E.dot(C.T),axis=0).mean()
        # checks if the prejected energy gets lower,
        # updates the minimum index in this case
        t = (i+1,j+1) if r < s else t
        s = r if r < s else s

      # updates the clusters' labels to the last
      # occurrence of t = (i_min,j_min), corresponding
      # to r_min. because we exclude the j_min - th
      self.c[self.c==t[1]]=t[0]
      self.c[self.c>t[1]] -= 1

      # updates centroids (excludes the j_min - th)
      # updates the projected spaces
      for j in range(t[1],self.k0):
        self.S[j] = self.S[j+1]
        self.E[j] = self.E[j+1]
      del self.E[self.k0],self.S[self.k0]

      # recalculates the projected space and centroid
      # of the newly merged cluster C_i' = C_i_min U C_j_min
      self.E[t[0]] = self.FindVectors(self.X[self.c==t[0]],l_new)
      self.S[t[0]] = np.mean(self.X[self.c==t[0]],axis=0)

      self.k0 -= 1

  def FindVectors(self,C,l):
  # FindVectors returns a matrix E whose rows are the l least
  # eigenvalued eigenvectors of the covariance matrix
  # of the Cluster inputed (as another matrix)
    U,S,V = np.linalg.svd(np.cov(C.T))
        #scipy.sparse.linalg.svds(np.cov(C.T))
    return V[-l:]
```

```

def Assign(self):
# self.S = {1:s_1,2:s_2,3:s_3, ... }
# self.E = {1:E_1,2E_2,3:E_3, ... }
# where s_i the i-th centroid and E_i is the i-th matrix whose
# rows are eigenvectors of each cluster If dist!='Euclid', then
# it should be given as dist = lambda x,y : dist_function(x - y)
# or something similar (dist = dist(x,y))

  dist = np.linalg.norm
  colors = []
  for x in self.X:
    z = np.array([ dist(self.E[i].dot(x - self.S[i])) for i in iter(self.S)])
    # color = cluster labels vector
    color = np.argmin(z)+1
    colors.append(color)
    self.c = np.array(colors)
  # After the first iteration, it starts to appear sometimes
  # empty clusters. sample2 will be needed if they happen to appear
  sample2 = random.sample(range(self.X.shape[0]),len(self.S))
  for i in iter(self.S):
    A = self.X[self.c==i]
    # when it happens to appear empty clusters, we force
    # them not to be so by randomly choosing one centroid
  if A.shape[0]==0:
    j = sample2[i-1]
    self.S[i] = self.X[j]
    self.c[j] = i
  elif A.shape[0] > 0:
    self.S[i] = np.mean(A,axis=0)
  else:
    print('strange thing happening, X.shape not 0 nor > 0 !')

  ################ end procedures ############
```

Bibliography

[1] N. M. Laird A. P. Dempster and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1):1–38, 1977.

[2] Charu C. Aggarwal and Philip S. Yu. Finding generalized projected clusters in high dimensional spaces. *SIGMOD Rec.*, 29(2):70–81, May 2000.

[3] Tom Angell.

[4] David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, SCG '06, pages 144–153, New York, NY, USA, 2006. ACM.

[5] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[6] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Advanced Applications in Pattern Recognition. Springer US, 2013.

[7] Gérard Biau and David DM Mason. High-dimensional p-norms. *arXiv preprint arXiv:1311.0587*, 2013.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[9] Avrim Blum. 15-859(b) machine learning theory, January 2010.

[10] Léon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. In *Advances in Neural Information Processing Systems 7*, pages 585–592. MIT Press, 1995.

[11] Sanjoy Dasgupta. Lecture 1 — Measure concentration, set 2006.

[12] Bernard Desgraupes.

[13] Chris Ding and Xiaofeng He. K-means clustering via principal component analysis. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 29–, New York, NY, USA, 2004. ACM.

[14] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming.* Springer International Publishing.

[15] Sariel Har-Peled and Bardia Sadri. How fast is the k-means method? In *SODA*, pages 877–885. SIAM, 2005.

[16] Chi Jin, Yuchen Zhang, Sivaraman Balakrishnan, Martin J. Wainwright, and Michael Jordan. Local Maxima in the Likelihood of Gaussian Mixture Models: Structural Results and Algorithmic Consequences, September 2016.

[17] I.T. Jolliffe. *Principal Component Analysis.* Springer Verlag, 1986.

[18] Leon S Lasdon. *Optimization theory for large systems.* Dover Books on Mathematics. Dover, Mineola, NY, 2011.

[19] S. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2):129–137, September 2006.

[20] Guojun Gan; Chaoqun Ma; and Jianhong Wu. *Data Clustering: Theory, Algorithms, and Applications.* ASA-SIAM Series on Statistics and Applied Probability. SIAM, Society for Industrial and Applied Mathematics, 2007.

[21] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms.* Cambridge University Press, New York, NY, USA, 2002.

[22] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[23] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In *Proceedings of the 3rd International Workshop on Algorithms and Computation*, WALCOM '09, pages 274–285, Berlin, Heidelberg, 2009. Springer-Verlag.

[24] C. M. Shetty Mokhtar S. Bazaraa, Hanif D. Sherali. *Nonlinear Programming: Theory and Algorithms.* Wiley-Interscience, 3rd edition, 2006.

[25] Andrew Ng. Stanford university cs229 lecture notes, November 2012.

[26] Andrew Ng. Stanford university cs229 lecture notes, November 2012.

[27] Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Chaitanya Swamy. The effectiveness of lloyd-type methods for the k-means problem. *J. ACM*, 59(6):28:1–28:22, January 2013.

[28] Varaiya P. *Lecture Notes on Optimization.* 1998.

[29] Dimitri Panteli Bertsekas. *Control of uncertain systems with a set-membership description of the uncertainty.* PhD thesis, Massachusetts Institute of Technology, 1971.

[30] R. Tyrrell Rockafellar. *Convex analysis.* Princeton Mathematical Series. Princeton University Press, Princeton, N. J., 1970.

[31] Jeffrey S. Rosenthal. *A first look at rigorous probability theory.* World Scientific, Singapore [u.a.], 2. ed edition, 2006.

[32] Shokri Z. Selim and M. A. Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(1):81–87, January 1984.

[33] Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference.* Springer Publishing Company, Incorporated, 2010.

[34] Wikipedia. Overfitting, 2017. [Online; accessed 12-june-2017].

[35] C. F. Jeff Wu. On the convergence properties of the em algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[36] K. Yeung and W. Ruzzo. An empirical study on principal component analysis for clustering gene expression data, 2001.

[37] W.I. Zangwill. *Nonlinear programming: a unified approach.* Prentice-Hall international series in management. Prentice-Hall, 1969.

[38] QINPEI ZHAO. *Cluster Validity in Clustering Methods.* PhD thesis, University of Eastern Finland, 2012.

[39] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Stat. Anal. Data Min.*, 5(5):363–387, October 2012.