Aprendizagem Automática

Assignment 2 - Clustering

Andrea Mascaretti N52222 Daniel Pimenta N45404

9th December 2017

Abstract

The purpose of this assignment was studying and learning the implementation of three major clustering algorithms: K-Means, DBSCAN and Gaussian Mixture Models and examine their individual performance in the problem of clustering seismic events.
To achieve this, we varied the main parameter of each algorithm (number of clusters for K-Means, neighbourhood distance ϵ for DBSCAN and number of gaussian components for Gaussian Mixture) and used an internal index (Silhouette score), and external indexes (Rand index, Precision, Recall, F1 measure, Adjusted Rand index, all computed from the fault line information) to examine the performance of all three clustering algorithms.
The data set used in this assigment was obtained from the USGS catalog and had the information about all seismic events of magnitude at least 6.5 in the last 100 years. In the end, we discussion of the advantages and disadvantages of each algorithm for this dataset, informed by an analysis of their behaviour with different parameters and the different scores previously referred. TODO brief summary of interpetations and conclusions

1 Introduction

1.1 A brief overview of clustering

Clustering is the process of examining a collection of "points", and grouping the points into "clusters" according to some distance measure. The goal is for points in the same clusters to have a small distance from one another, while points in different clusters are at a large distance from one another ([LRU14]). In this sense, clustering is helpful when it comes to understanding the structure of the data at hands, especially when dealing with high dimensional datasets. Clustering can provide us with a representation of the data that can facilitate gaining some meaningful insight into their structure. Moreover, it can be applied to reduce complexity and summarise data when this may be needed.

A dataset suitable for clustering is a collection of points $\{\boldsymbol{x}_i\}_{i=1}^N \subset \mathcal{X}$. In a general setting, $\mathcal{X}$ can be any abstract space, but it is often the case that it is a Euclidean space. To perform clustering we need to endow $\mathcal{X}$ with a distance measure (or metric) δ. Note that a metric on a set $\mathcal{X}$ is a function $\delta : \mathcal{X} \times \mathcal{X} \to [0, +\infty)$ such that for all $x, y, z \in \mathcal{X}$

1. $\delta(\boldsymbol{x}, \boldsymbol{y}) \geq 0$ (non-negativity)
2. $\delta(\boldsymbol{x}, \boldsymbol{y}) = 0 \iff x = y$ (identity of indiscernibles)
3. $\delta(\boldsymbol{x}, \boldsymbol{y}) = \delta(\boldsymbol{y}, \boldsymbol{x})$ (symmetry)
4. $\delta(\boldsymbol{x}, \boldsymbol{z}) \leq \delta(\boldsymbol{x}, \boldsymbol{y}) + \delta(\boldsymbol{y}, \boldsymbol{z})$ (triangular inequality)

Clustering algorithms can be differentiated (some may say *clustered*) into two main different categories, according to the strategy implemented.

Hierarchical clustering In hierarchical clustering, every point is initially assigned to its own cluster. Clusters are then merged according to their "closeness", where we can use one of

the many definitions of the term. The process is ended according to some stopping criteria, for example because we have reached a given number of total cluster input as a threshold or because from merging any of the given clusters we obtain a situation in which data points are too "spread out" according to the definition of "closeness" in use.

Partitional clustering In partitional clustering, points are considered in some order and assigned iteratively to the clusters to which the fit best. This phase usually follows another one, during which initial cluster are estimated. Different algorithms in this category allow combining or splitting clusters, or leaving points unassigned when they are *outliers* (i.e., points too far from any cluster).

Other two possible classification frameworks, that we mention only briefly, consider other properties:

(a) We have algorithms that assume that $\mathcal{X}$ is a Euclidean space and others that only consider $(\mathcal{X}, \delta)$ to be an abstract measure space. In the first case, it is possible to summarise a collection of points by means of a *centroid* - a notion we will introduce later - whereas in the second there is no such thing and we are forced to find a different way to summarise the information contained within a cluster.

(b) In this case, the main assumption regards the memory of the machine we are working on. We have two possibilities: we can either assume that the data set is small enough to fit into the main memory or that our data resides in some secondary memory. Sometimes it is infeasible to look at all the pairs of data and strategies must be developed to respond to that. Another problem arising is that of summarising the information contained in every cluster into the main memory for computations.

The clustering algorithms considered in this work, namely the K-Means, the DBSCAN and the Gaussian mixture models, are partitional clustering algorithms working in Euclidean spaces. What is more, given the dimensionality of our dataset we will be working on the main memory of the computer, thus avoiding all the issues caused by highly-dimensional data.

1.2 Clustering Validation

Given the wide variety of existing clustering methods, it is of vital importance to develop objective approaches to validate and assess clustering results. In this sense, three different tasks arise naturally: *clustering evaluation* seeks to estimate the quality of the clustering, *clustering stability* seeks to assess how sensitive a given algorithm is with respect to its parameters and *clustering tendency* aims at understanding whether it is the case to apply clustering in the first place ([ZWM14]). We can divide the numerous validity measures and statistics into three main categories.

External External validation measures employ criteria that are not inherently present in the dataset. For instance, we can have prior or expert-specified clusters.

Internal Internal validation measures employ criteria that are based on the data provided for the clustering, such as measure of intra-cluster and inter-cluster dispersion.

Relative Relative validation measures aim to directly compare different types of clusterings, usually obtained setting the parameters of one given algorithms in different ways.

We hereby introduce five external indices, precision, recall, F1-score, the Rand index and the adjusted Rand index, and one internal index, the silhouette coefficient. Before, however, we will require some notations. Let $\mathcal{D} = \{\boldsymbol{x}_j\}_{j=1}^n \subset \mathcal{X}$ be a dataset containing n points such that the dimension of $\mathcal{X}$ is equal to d, which we have partitioned into k clusters. Let $y_i \in \{1, 2, ..., k\}$ denote the ground-truth cluster membership (or label information) for each point. The ground-truth clustering can be expressed as $\mathcal{T} = \{T_1, T_2, ..., T_k\}$, where $T_j = \{\boldsymbol{x}_i \in \mathcal{D} | y_i = j\}$. Moreover, let $\mathcal{C} = \{C_1, ..., C_r\}$ denote a clustering on the same dataset into r clusters, obtained by means of a certain clustering algorithm. We let $\hat{y}_i \in \{1, 2, ..., r\}$ denote the cluster label for $\boldsymbol{x}_i$. For the sake of clarity, we shall refer henceforth to $\mathcal{T}$ as *partitioning* and to each T_i as *partition*. In

a similar fashion, $\mathcal{C}$ will be denoted as a clustering and each C_i as a cluster. External validation measures try to capture the extent to which points belonging to the same partition appear in the same clusters and, of course, the extent to which points belonging to different partitions are grouped into different clusters. Note that there is usually a trade-off between these two goals, either captured by a measure or implicitly present in its computation. External partition relies on the $r \times k$ *contingency table* N that is induced by the clustering $\mathcal{C}$ and the ground-truth partitioning $\mathcal{T}$ defined as follows:

$$N = \{n_{ij}\}$$

where

$$n_{ij} = |C_i \cap T_j|.$$

In other words, the count n_{ij} denotes the number of points that are common to cluster C_i and to the ground-truth partition T_j. Further, we let $n_i = |C_i|$ and $m_j = |T_j|$. Let then $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{D}$ be any two points such that $i \neq j$. If both $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same cluster, or, equivalently, if $\hat{y}_i = \hat{y}_j$, we call it a *positive* event and if they do not belong to the same cluster, that is if $\hat{y}_i \neq \hat{y}_j$, we call that a *negative* event. Depending on whether there is agreement between the cluster labels and the partition labels, we thus have four possibilities to consider.

True Positives $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong both to the same partition in $\mathcal{T}$ and to the same cluster in $\mathcal{C}$. We have a true positive pair because the positive event correspond to the ground truth.

$$TP = |\{(\boldsymbol{x}_i, \boldsymbol{x}_j) : y_i = y_j, \hat{y}_i = \hat{y}_j\}|$$

False Negatives $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same partition in $\mathcal{T}$ but to different clusters in $\mathcal{C}$. Thus, the negative event does not correspond to the truth.

$$FN = |\{(\boldsymbol{x}_i, \boldsymbol{x}_j) : y_i = y_j, \hat{y}_i \neq \hat{y}_j\}|$$

False Positives $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ do not belong to the same partition in $\mathcal{T}$ but they do belong to the same cluster in $\mathcal{C}$. The positive event is therefore false.

$$FP = |\{(\boldsymbol{x}_i, \boldsymbol{x}_j) : y_i \neq y_j, \hat{y}_i = \hat{y}_j\}|$$

True Negative $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ neither belong to the same partition in $\mathcal{T}$ nor to the same cluster in $\mathcal{C}$. The false event is therefore true.

$$TN = |\{(\boldsymbol{x}_i, \boldsymbol{x}_j) : y_i \neq y_j, \hat{y}_i \neq \hat{y}_j\}|$$

The total pair of points is given by $N = \frac{n(n-1)}{2}$ and we have the following identity:

$$N = TP + FN + FP + TN.$$

A naive computation of the preceding four cases requires $O\left(n^2\right)$ time. However, there is a more efficient way to proceed with the calculations, considering the contingency table $N = \{n_{ij}\}$ with $1 \leq i \leq r$ and $1 \leq j \leq k$. The number of true positives is then

$$TP = \sum_{i=1}^{r}\sum_{j=1}^{k}\binom{n_{ij}}{2} = \sum_{i=1}^{r}\sum_{j=1}^{k}\frac{n_{ij}\left(n_{ij}-1\right)}{2} = \frac{1}{2}\left(\sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2 - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}\right) = \frac{1}{2}\left(\left(\sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2\right) - n\right).$$

This follows from the fact that each pair of points among the n_{ij} share the same cluster label i and the same partition label j and of course the last step follows from the fact that $\sum_{i=1}^{r}\sum_{j=1}^{k} = n$. To compute the total number of false negatives, we have to remove the number of true positives from the number of pairs that belong to the same partition. Because two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ that belong to the same partition have $y_i = y_j$, if we remove true positives we are left with pairs for which $\hat{y}_i \neq \hat{y}_j$, which is exactly the false negatives. This yields

$$FN = \sum_{j=1}^{k}\binom{m_{ij}}{2} - TP = \frac{1}{2}\left(\sum_{j=1}^{k}m_j^2 - \sum_{j=1}^{k}m_j - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2 + n\right) = \frac{1}{2}\left(\sum_{j=1}^{k}m_j^2 - \sum_{i=1}^{r}\sum_{j=1}^{k}n_{ij}^2\right).$$

The number of false positives can be obtained in a very similar fashion:

$$FP = \sum_{i=1}^{r} \binom{n_i}{2} - TP = \frac{1}{2}\left(\sum_{i=1}^{r} n_i^2 - \sum_{i=1}^{r}\sum_{j=1}^{k} n_{ij}^2\right).$$

To conclude, we can find the true negatives by doing

$$TN = N - (TP + FN + FP) = \frac{1}{2}\left(n^2 - \sum_{i=1}^{r} n_i^2 - \sum_{j=1}^{k} m_j^2 + \sum_{i=1}^{r}\sum_{j=1}^{k} n_{ij}^2\right).$$

In this way, we can compute the values in $O(rk)$ time. As the contingency table can be obtained in linear time, the total time is $O(n + rk)$. Having said that, we can proceed to define the external indices we shall be using throughout our analysis.

Precision Precision, in the context of information retrieval, is defined as the fraction of retrieved documents that are relevant. We define it by analogy here as

$$\frac{TP}{TP + FP},$$

obtaining an index that gives us the fraction of true, or correctly classified, point pairs compared to all the point pairs in the same clusters. Note that the index will always be contained in $[0, 1]$.

Recall Recall, in the context of information retrieval, is defined as the fraction of relevant documents that are actually retrieved. As done before, we define it by analogy

$$\frac{TP}{TP + FN}.$$

Here we measure the fraction of correctly labelled points pairs compared to all the points pairs in the same partition. Note that the index will always be contained in $[0, 1]$.

F1-Score The F1-Score is defined to be the harmonic mean between precision and recall:

$$2\frac{precision \cdot recall}{precision + recall}$$

When both recall and precision are perfect, the value is 1 and is bounded below by 0.

Rand Statistics The Rand statistics measures the fraction of true positive and true negatives over all point pairs. It is defined as

$$Rand = \frac{TP + TN}{N}.$$

The Rand statistics measures the fraction of point pairs where both $\mathcal{C}$ and $\mathcal{T}$ agree. A perfect clustering has a value of 1 for the statistics.

Adjusted Rand Index The adjusted Rand index builds on the Rand index, correcting it for chance. It can yield negative values and it is defined as

$$AdjustedRandIndex = \frac{\sum_{i=1}^{r}\sum_{j=1}^{k}\binom{n_{ij}}{2} - \frac{\sum_{i=1}^{r}\binom{n_i}{2}\sum_{j=1}^{k}\binom{m_j}{2}}{\binom{n}{2}}}{\frac{1}{2}\left[\sum_{i=1}^{r}\binom{n_i}{2} + \sum_{j=1}^{k}\binom{m_j}{2}\right] - \frac{\left[\sum_{i=1}^{r}\binom{n_i}{2}\sum_{j=1}^{k}\binom{m_j}{2}\right]}{\binom{n}{2}}}$$

We now move on to describing the internal index we are going to use.

Silhouette Coefficient The silhouette coefficient is a measure of cohesion and separation of clusters. It is based on the difference between the average distance to points in the closest cluster and to points in the same clusters. For each point $\boldsymbol{x}_i$, we calculate its silhouette coefficient s_i as

$$s_i = \frac{\mu_{out}^{\min}(\boldsymbol{x}_i) - \mu_{in}(\boldsymbol{x}_i)}{\max\left\{\mu_{out}^{\min}(\boldsymbol{x}_i), \mu_{in}(\boldsymbol{x}_i)\right\}},$$

where $\mu_{in}(\boldsymbol{x}_i)$ is the mean distance from $\boldsymbol{x}_i$ to points in its own cluster $\hat{y}_i$:

$$\mu_{in}(\boldsymbol{x}_i) = \frac{\sum_{\boldsymbol{x}_j \in C_{\hat{y}_i}, j \neq i} \delta(\boldsymbol{x}_i, \boldsymbol{x}_j)}{n_{\hat{y}_i} - 1}$$

and

$$\mu_{out}^{\min}(\boldsymbol{x}_i) = \min_{j \neq \hat{y}_i} \left\{ \frac{\sum_{\boldsymbol{y} \in C_j} \delta(\boldsymbol{x}_i, \boldsymbol{y})}{n_j} \right\}$$

The s_i value of a point always lies in the interval $[-1, 1]$. A value closer to $+1$ indicates that $\boldsymbol{x}_i$ is much closer to points in its own cluster than to points in other clusters. A value close to zero indicates that $\boldsymbol{x}_i$ is close to the boundary and a value close to -1 indicates that $\boldsymbol{x}_i$ is closer to another cluster than it is to its own. The silhouette coefficient is defined as

$$SC = \frac{1}{n} \sum_{i=1}^{n} s_i$$

and, as it is natural, a value close to $+1$ indicates a good clustering.

1.3 The USGS Catalog

The dataset on which we shall develop our analysis was obtained from the USGS catalog. It comprises all the geographical coordinates of every seismic event of a magnitude of at least 6.5 occurred during the past century. A column, specifying the fault line between the tectonic plates that are closest to the seismic event was also considered. It was obtained from the same source and it will serve the role of providing a ground-truth partitioning for external validation. Each fault line is identified by an non-negative integer, with the exception of those earthquakes that were not significantly close to any fault line: they are labelled as -1. Before proceeding with the analysis, we transformed the longitude and latitude values into Earth-centred, Earth-fixed coordinates (x, y, z) as follows:

$$x = R \cos\left(latitude \cdot \frac{\pi}{180}\right) \cos\left(longitude \cdot \frac{\pi}{180}\right)$$
$$y = R \cos\left(latitude \cdot \frac{\pi}{180}\right) \sin\left(longitude \cdot \frac{\pi}{180}\right)$$
$$z = R \sin\left(latitude \cdot \frac{\pi}{180}\right),$$

where $R = 6731\,\text{Km}$, being the average radius of the Earth. This was done as necessary to compute distances between two points. Apart from that, no other transformation was added to the data: we did not scale them, as all the axes are measured according to the same unit (namely kilometres).

The analysis was conducted using the Python programming language. In particular, the Scikit-Learn package ([PVG+11]), the NumPy package ([vdWCV11]) and the Matplotlib package ([Hun07]) were used.

2 Clustering Algorithms

2.1 K-Means Algorithm

In our exposition of the K-Means Algorithm we rely heavily on [HTF01], [JW07] and [ZWM14]. Suppose that we have a set of n observations, indexed by $I = \{1, ..., n\}$, of some quantitative variables

$$C = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$$

where

$$\boldsymbol{x}_i \in \mathbb{R}^p,\ \forall i \in I$$

We also have a set of labels (at this stage we will consider their number as given and such that $card(\mathcal{L}) = k,\ k \in [1, n] \cap \mathbb{N}$)

$$\mathcal{L} = \{l_1, \ldots, l_k\},\ K := \{1, \ldots, k\}$$

Our goal is to construct a (*measurable*) function to associate each and every element of C to one label in $\mathcal{L}$. It is easy to see that it makes no difference to work with $\mathcal{L}$ or K as we can always find a bijection between the two sets.

We have reasons to believe that the elements belonging to one particular group are somewhat more similar to each other than they are to members of other groups and that one element can belong to one group and one group only. We also hypothesise that there are no empty groups. Translating this into mathematical terms, our aim is to find a collection

$$\{C_i\}_{i \in K}$$

such that

$$C_i \cap C_j = \emptyset,\ i \neq j$$

and

$$\bigcup_{i=1}^{k} C_i = C$$

or, in other words, we want to partition C into k clusters.

Moreover, for each cluster we wish to find a representative member to summarise the information contained within the cluster. An usual choice is to consider the mean, or **centroid**, of the cluster defined as

$$\boldsymbol{c}_i = \frac{1}{n_i} \sum_{\boldsymbol{x}_j \in C_i} \boldsymbol{x}_j,$$

where n_i is the cardinality of C_i.

The main issue is to find a method to partition the set C into the clusters. A brute-force algorithm for finding a good clustering would imply generating all the possible partition of n points into k cluster and then, once an evaluation function has been set, take the best one. Informally, however, we can see that each point can be assigned to any of the k clusters so that there are at most k^n possible clusterings. Considering that any permutation of the k clusters within a given clustering yields an equivalent result, we see that there are $O(\frac{k^n}{k!})$ clusterings of n points into k clusters. It is therefore clear that we need to rely on a different strategy.

First of all, we need to define in a more precise manner the meaning of similarity between two points: we will consider the squared *Euclidean distance* for this purpose

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{k=1}^{p} (x_{ik} - x_{jk})^2 .$$

We now need to define some *loss* function and try to minimise it. The most natural idea follows from the assumption that members of one given cluster will be closer to each other than to member of other clusters: this is equivalent to stating that the correct partitioning will minimise the variability *within* the clusters. We need to translate this key concept into an objective function.

We define

$$I_k := \{i \in I\ s.t.\ \boldsymbol{x}_i \in C_k\}$$

and in this case we will have

$$W(C_k) = \frac{1}{n_k} \sum_{i,j \in I_k} \sum_{l=1}^{p} (x_{il} - x_{jl})^2$$

and the problem becomes as follows [1]

$$\min_{C_1,\dots,C_k} \left\{ \sum_{k=1}^{K} W(C_k) \right\}$$

To solve the problem, K-Means applies a greedy iterative algorithm to find a local optimum.

The centroids are randomly generated into the data space, usually considering a uniform distribution between the range for each dimension. Every iteration, then, consists of two main steps: (1) cluster assignment and (2) centroid update. That is to say, given the k cluster means, in the cluster assignment step, each point $\boldsymbol{x}_j$ is assigned to the closest mean, which induces a clustering (each cluster C_i comprises all the points that are closer to $\boldsymbol{c}_i$ than to any other centroid).

$$i^* = \arg\min_{i \in K} \{d(\boldsymbol{x}_j, \boldsymbol{c}_i)\} = \arg\min_{i \in K} \{d(\boldsymbol{x}_j, \boldsymbol{c}_1), \dots, d(\boldsymbol{x}_j, \boldsymbol{c}_k)\} \implies \boldsymbol{x}_j \in C_{i^*}$$

Now, given a set of cluster C_i, $i = 1, ..., k$, in the centroid update step new values are computed for each cluster from the points in C_i. The cluster assignment and centroid update steps are repeated until we reach a fixed point or a local minimum. In more practical words, we can say that K-Means converges if the centroids do not change from one iteration to the next and we can stop when the distance between an iteration and another is smaller than some positive threshold ε.

The algorithm we have just defined is descent, that is it improves the solution at each iteration so that when the termination condition is true, we are certain to have reached a *local optimum*. Considering also that we randomly set the initial centroids, it is a good practice to run the algorithm several time and report the clustering with the evaluation value. It is also worth noting that K-Means generates convex-shaped clusters as the region in the data space corresponding to each cluster can be obtained intersecting the half-spaces resulting from the hyperplanes that bisect and are normal to the segments that join pairs of centroids.

There is also one last important point to state. Any time we perform the K-means methods, we will find k clusters on our data. Now, the question is: are we really separating actual subgroups or are we just *clustering the noise*? Moreover, suppose that we are dealing with a case in which the vast majority of data belongs to a small number of subgroups, but for a small subset that is quite different from the rest. As our method *forces* our data into a fixed number of clusters, the presence of an outlying class might bring to distorted results. So it is good practice not only to try various values of k to see what happens, but also to try and apply to algorithm to subsets of our initial set to check that results remain somewhat stable.

2.1.1 Algorithm for K-means Clustering

1. Randomly assign a number, from 1 to K, to each observation: they will serve as initial cluster assignments. Set $\mu := 0$
2. Iterate what follows until termination criterion holds:
   (a) For each cluster, compute the *centroid*

   $$\boldsymbol{c}_i^{\mu} = \frac{1}{card(C_i^{\mu})} \sum_{j \in I_i^{\mu}} \boldsymbol{x}_i,$$

   computed as the mean vector of the observations in the k-th cluster at μ-th iteration
   (b) Assign each observation to the cluster whose centroid is closest (with respect to the Euclidean distance) as follows

   $$\forall j \in I,\ i^* = \min_{i \in K} \left\{ d(\underline{x}_j, \underline{c}_i^{\mu}) \right\} = \min \left\{ d(\underline{x}_j, \underline{c}_1^{\mu}), \dots, d(\underline{x}_j, \underline{c}_k^{\mu}) \right\} \implies \underline{x}_j \in C_{i^*}^{\mu}$$

   (c) Set $\mu := \mu + 1$
3. Termination occurs when cluster assignments stop changing

[1] It is important to notice that this function surely decreases for increasing k and in the extreme case of $k = n$ we have that each cluster contains only one point and we have no variability whatsoever. It is important to understand that we are looking for classes that do have some internal variability.

2.1.2 Choosing the number of clusters

In order to choice the number of clusters that seemed to yield the best results, we decided to run the algorithm for different values of k, from 1 to 100. The results are shown in Figure 1. First of all, we can point out that there appear to be an inverse correlation between precision and recall, as expected. In facts, as the number of clusters grows we have an increased rate of false negatives. This is given to the fact that the increased number of clusters makes it more likely to have points belonging to the same partition that are classified into different clusters. On the other hand, increasing k implies a decrease in false positives as it is more unlikely that points belonging to the same partition are clustered into different clusters. The trade-off is also evident considering the F1-Score, that performs better when we have high recall and a good value for the precision (that could anyway improve allowing for greater k). It is interesting to see that the number of clusters suggested by precision and recall are very different. In this sense, the best choice depends very much on what it is important to maximise: whether it is the fraction of positive events that are true compared to the point in the same cluster or the fraction of correctly labelled point within a cluster with respect to al the points pairs in the same partition. The Rand index and the adjuster Rand index suggests a similar number of clusters (if we consider the number that we obtain considering the "elbow" in the Rand index plot). However, we see how the adjusted Rand index deteriorates when we pass that number: it is due to the correction for chance done by the index. As far as the Silhouette coefficient is concerned, we see that we obtain again a suggested value between 20 and 40, where we have a good balance between cohesion and separation of clusters. We obtain a result that substantially agree with the last three external indices. This may be due to the intrinsic nature of the K-Means algorithm, that has to rely on convex-shaped clusters and cannot therefore adapt too much to the domain at hands, characterised by rather "prolonged" clusters in shape (as it is often the case with geophysical data): with a small number of clusters we have a good recall, that we pay in terms of precision, and as soon as we try to augment it we see that we lose in terms of fidelity to the original partitioning. We provide a representation of the clustering induced (Figure 2) considering the best value obtained by means of the Silhouette score, which should not be too different from the clustering induced by the F1-Score, the Rand index and the adjusted Rand index as already explained. In Table 1, we report the indices we obtain maximising for each of them.

Index	Precision	Recall	F1-Score	Rand Index	Adjusted Rand Index	Silhouette Score	k
Prec.	0.761	0.093	0.166	0.502	0.147	0.496	146
Recall	0.127	0.791	0.219	0.326	0.056	0.481	2
F1-Sc.	0.347	0.667	0.457	0.483	0.374	0.374	6
Rand Ind.	0.619	0.244	0.350	0.503	0.311	0.516	39
Adj. Rand Ind.	0.575	0.352	0.437	0.503	0.390	0.507	22
Silh. Score	0.524	0.299	0.380	0.501	0.332	0.528	25

Table 1: Values of the Precision, Recall, F1-Score, Rand index, adjusted Rand index and Silhouette score at the k maximising the index considered in the row.

2.2 DBSCAN

In our analysis of the DBSCAN algorithm, we shall be relying mainly on [EKSX96] and [ZWM14]. The necessity of an algorithm adopting a different philosophy with respect to that of the K-Means algorithm arises from the fact that representative methods (such as the latter or, for instance, EM methods) are suitable for finding, at best, convex clusters. However, for non-convex clusters these methods are not capable to find the true clusters. This is because two points belonging to two different clusters might actually be close to each others than two points from the same on. For this reason, density-based clustering algorithms adopt a different paradigm, using the local densities of points to determine the clusters rather than relying solely on distances between couple of them.

We now introduce some preliminary concepts. First of all, we define a **ball** of radius ε around a point $\boldsymbol{x} \in \mathbb{R}$, called *$\varepsilon$-neighbourhood* of $\boldsymbol{x}$, as follows:

$$\mathcal{N}_\varepsilon(\boldsymbol{x}) = B_\delta(\boldsymbol{x}, \varepsilon) = \{\boldsymbol{y} \; s.t. \; \delta(\boldsymbol{x}, \boldsymbol{y}) \leq \varepsilon\}.$$

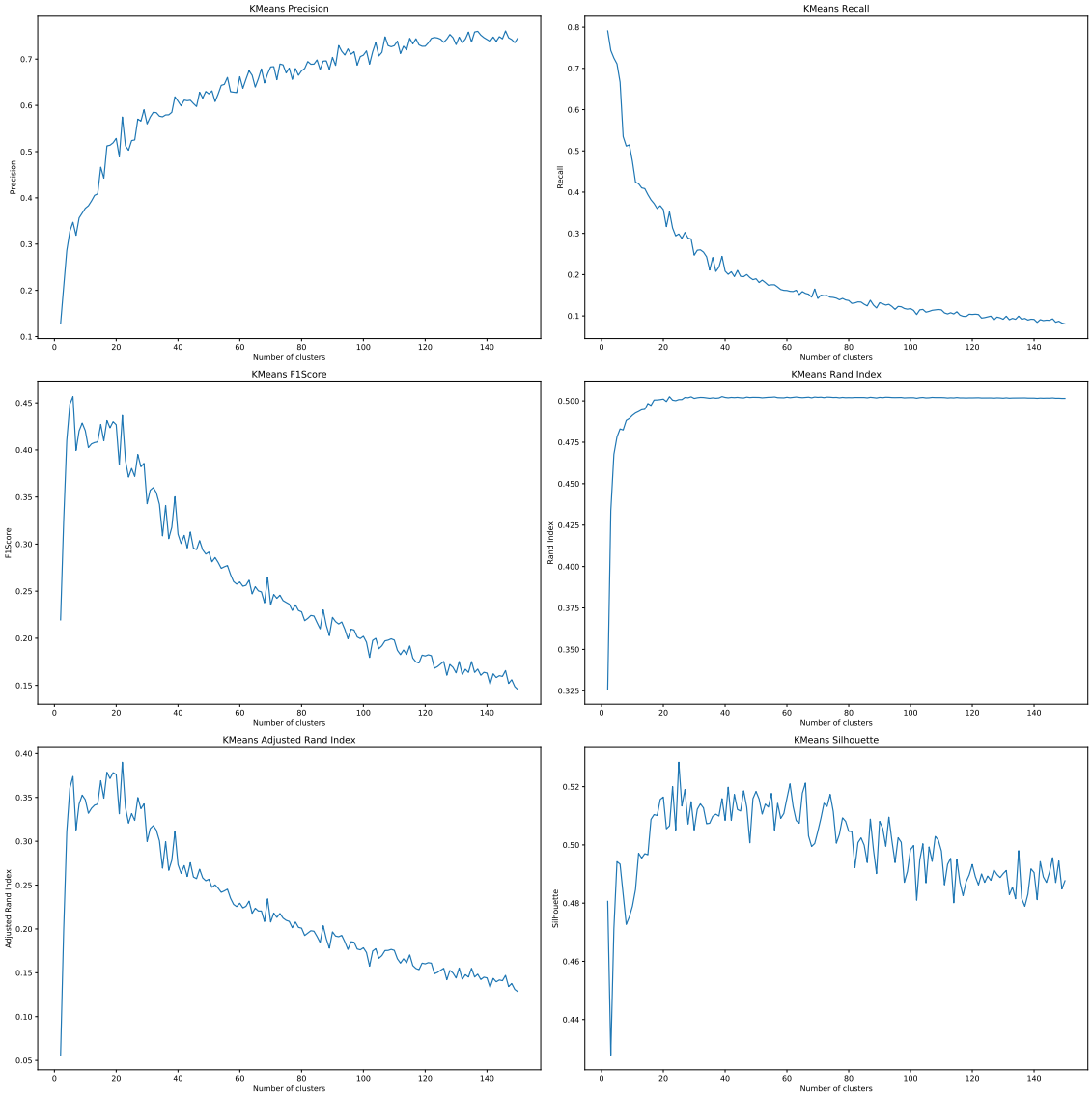

Figure 1: Precision, Recall, F1-Score, Rand Index, Adjusted Rand Index and Silhouette Score for the K-Means algorithms at different values of k.

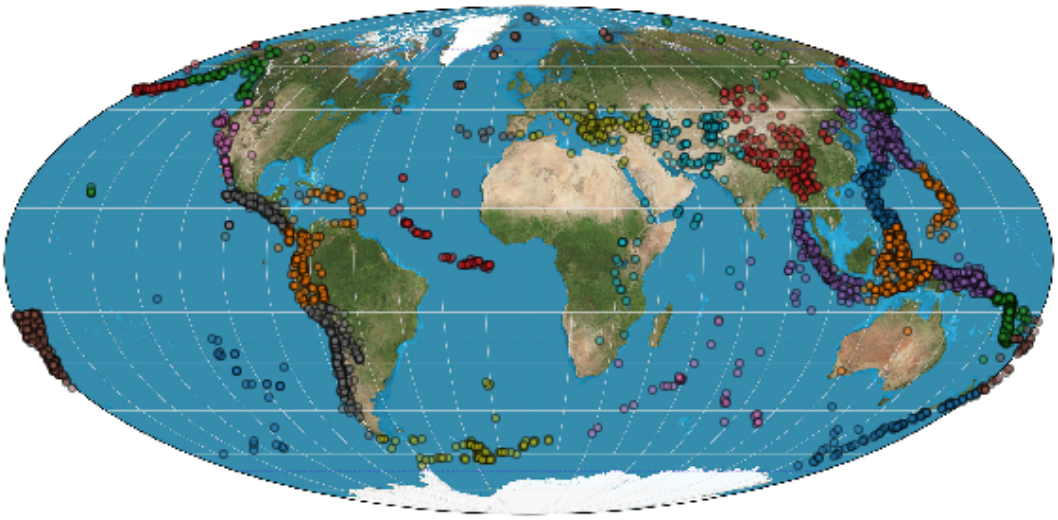

Figure 2: Representation of the clustering obtained applying the K-Means algorithm with $k = 25$.

Here $\delta(\boldsymbol{x}, \boldsymbol{y})$ represents the distance between the points $\boldsymbol{x}$ and $\boldsymbol{y}$, which is usually assumed to be the Euclidean distance, that is

$$\delta(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}||_2.$$

However, other distance metrics can be used. For any point $\boldsymbol{x} \in \boldsymbol{D}$, where $\boldsymbol{D}$ is the set of data points, we say that $\boldsymbol{x}$ is a *core point* if there are at least k points in its ε-neighbourhood. In other words, $\boldsymbol{x}$ is defined as a core point when it holds:

$$|\mathcal{N}_\varepsilon(\boldsymbol{x})| \geq k,$$

where k is a local density (or frequency) threshold defined by the user.

A *border point* is defined as a point that does not meet the k threshold or, equivalently, that is such that $|\mathcal{N}_\varepsilon(\boldsymbol{x})| < k$, but belongs to the ε-neighbourhood of some core point $\boldsymbol{z}$. Finally, if a point is neither a core point nor a border point, it is called a *noise point* or outlier. We say that a point $\boldsymbol{x}$ is *directly density reachable* from another point $\boldsymbol{y}$ if $\boldsymbol{x} \in \mathcal{N}_\varepsilon(\boldsymbol{y})$ and $\boldsymbol{y}$ is a core point. We say that $\boldsymbol{x}$ is *density reachable* from $\boldsymbol{y}$ if there exists a chain of point, $\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_l$, such that $\boldsymbol{x} = \boldsymbol{x}_0$ and $\boldsymbol{y} = \boldsymbol{x}_l$, and $\boldsymbol{x}_i$ is directly density reachable from $\boldsymbol{x}_{i-1}$ for all $i = 1, ..., l$. In other words, there must be a set of core points leading from $\boldsymbol{y}$ to $\boldsymbol{x}$. Note that the relationship of density reachability is an asymmetric property (or a directed relationship). Define any two points $\boldsymbol{x}$ and $\boldsymbol{y}$ to be density connected if there exists a core point $\boldsymbol{z}$, such that both $\boldsymbol{x}$ and $\boldsymbol{y}$ are density reachable from $\boldsymbol{z}$. A *density based cluster* is defined as a maximal set of density connected points.

The algorithm works as follows. First, DBSCAN computes the ε-neighbourhood $\mathcal{N}_\varepsilon(\boldsymbol{x}_i)$ for each point $\boldsymbol{x}_i \in \boldsymbol{D}$. It then checks if $\boldsymbol{x}_i$ is a core point. Moreover, it sets the cluster id $id(\boldsymbol{x}_i) = \emptyset$ for all points, indicating that they are not assigned to any cluster. Next, starting from each core point that is not assigned, the algorithm recursively finds all its density connected points, which are assigned to the same clusters. Some border point may be reachable from core points in more than on cluster: they may either be arbitrarily assigned to one of the clusters or to all of them (if overlapping clusters are allowed!). Those points that do not belong to any cluster are treated as outliers or noise. DBSCAN can be viewed from another point of view. In fact, we can see it as a search for connected components in a graph with the following characteristics: the vertices corresponds to the core points in the dataset and there exists an (undirected) edge between two vertices (core points) if the distance between them is less than ε, that is to say, if each of them is in the ε-neighbourhood of the other point. The connected components of this graph correspond to the core points of each cluster. Next, every core point incorporates into its cluster any border points in its neighbourhood. The biggest drawback of DBSCAN is given by its sensitivity to the

value of ε, in particular if clusters have different densities. What happens is that is ε is too small, than sparser clusters will be categorised as noise and if it is too large different clusters might end up being grouped together.

2.3 Gaussian Mixture

In order to introduce Gaussian Mixture models, we shall be relying mainly on [Bis06]. A Gaussian mixture distribution can be written as a linear superposition of Gaussians in the form

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k\right) \tag{1}$$

However, it is possible to rewrite (1) as follows. Let us a introduce a K-dimensional binary random variable $\boldsymbol{z}$ having a 1-of-K representation so that a particular element $z_k = 1$ and all the other elements are equal to 0. We have that $\sum_k z_k = 1$ and that, for all k, $z_k \in \{0, 1\}$. We can see that there are K possible states for which no element of $\boldsymbol{z}$ is zero. We now proceed to defining the joint distribution of $p(\boldsymbol{x}, \boldsymbol{z})$ as $p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$. We set

$$p(z_k = 1) = \pi_k,$$

with $\pi_k \in [0, 1]$ and $\sum_{k=1}^{K} \pi_k = 1$, which we can rewrite

$$p(\boldsymbol{z}) = \prod_{k=1}^{K} \pi_k^{z_k}.$$

Similarly, we the conditional distribution of $\boldsymbol{x}$ given $\boldsymbol{z}$ is

$$p(\boldsymbol{x}|z_k = 1) = \mathcal{N}\left(\boldsymbol{x}|\boldsymbol{\mu}, \Sigma_k\right).$$

We can rewrite the last equation as

$$p(\boldsymbol{x}|\boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \Sigma_k)^{z_k}.$$

The conditional distribution of $\boldsymbol{z}$ given $\boldsymbol{x}$ is computed considering the theorem of Bayes. Firstly, we let $\gamma(z_k) = p(z_k = 1|\boldsymbol{x})$. We have than

$$\gamma(z_k) = \frac{p(z_k = 1)p(\boldsymbol{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\boldsymbol{x}|z_j = 1)} = \frac{\pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \Sigma_j)} \tag{2}$$

where, if π_k is the prior, we have that $\gamma(\pi_k)$ is the posterior. To put in another way, we can see that $\gamma(\pi_k)$ is the *responsibility* component k takes for explaining observation $\boldsymbol{x}$. Suppose, then, that we have a set of observations $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, and we wish to model this data using a mixture of Gaussians. We adopt a matrix representation, so that we have a matrix X in which the n^{th} row is given by $\boldsymbol{x}_n^T$. In a similar fashion we denote the $N \times K$ matrix Z. Assuming that the data points are independent and identically distributed from the distribution, we can write the log-likelihood function as

$$\ln p\left(X|\boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma\right) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \Sigma_k\right) \right\} \tag{3}$$

Maximising this function yields many problems, due to the presence of singularities. In facts, suppose that component j has mean $\boldsymbol{\mu}_j$, exactly equal to the data point $\boldsymbol{x}_n$ in the data given. This point will contribute to likelihood function as

$$\mathcal{N}\left(\boldsymbol{x}_n|\boldsymbol{x}_n, \sigma^2\mathcal{I}\right) = \frac{1}{(2\pi)^{\frac{1}{2}}} \frac{1}{\sigma_j}$$

where we have the simplifying hypothesis $\Sigma_k = \sigma_k^2 \mathcal{I}$. It is clear that the limit for $\sigma_j \to 0$ goes to infinity, and so does the log-likelihood. Thus, the problem is not well posed as such singularities will always be present and will occur whenever one of the Gaussian components collapses on one specific data point. A further issue is given by the fact that for any given maximum likelihood solution, a K-component mixture will have a total of $K!$ equivalent solutions. Each of them corresponds to the $K!$ possible ways of assigning K sets of parameters to K components.

2.3.1 EM for Gaussian mixtures

To find a maximum likelihood solution for our problem, we shall rely on an *expectation-maximisation* algorithm, or EM algorithm. To begin we want to write down the stationarity conditions that must be satisfied by the log-likelihood function. Setting the derivative with respect to $\boldsymbol{\mu}_k$ of (3) to zero, we obtain

$$0 = -\sum_{n=1}^{N} \frac{\pi_k \mathcal{N}\left(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \Sigma_k\right)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x}_n | \boldsymbol{\mu}_j, \Sigma_j\right)} \Sigma_k \left(\boldsymbol{x}_n - \boldsymbol{\mu}_k\right)$$

Assuming that Σ_k is invertible and remembering (2), we can multiply by Σ_k^{-1} and rearranging to obtain

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{k=1}^{K} \gamma\left(z_{nk}\right) \boldsymbol{x}_n,$$

where we have defined

$$N_k = \sum_{n=1}^{N} \gamma\left(z_{nk}\right)$$

We can interpret N_k as the effective number of points assigned to cluster k. We also see that the mean $\boldsymbol{\mu}_k$ for the k^{th} Gaussian components is obtained by taking a weighted mean of all the points of the data set, in which the weight is given by the posterior probability $\gamma\left(z_{nk}\right)$ that component k was responsible for generating $\boldsymbol{x}_n$.

If we set the derivative of (3) with respect t Σ_k to zero and follow a similar line of reasoning, we obtain

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma\left(z_{nk}\right) \left(\boldsymbol{x}_n - \boldsymbol{\mu}_k\right) \left(\boldsymbol{x}_n - \boldsymbol{\mu}_k\right)^T,$$

which has the same form as the corresponding result for a single Gaussian but with each data point weighted according to its posterior probability. Finally, we maximise (3) with respect to the mixing coefficients π_k. Considering that they must sum to one, we have to maximise (introducing Lagrange multipliers)

$$\ln p\left(X | \boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma\right) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1\right),$$

which gives

$$0 = \sum_{n=1}^{N} \frac{\pi_k \mathcal{N}\left(\boldsymbol{x}_n | \boldsymbol{\mu}_k, \Sigma_k\right)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x}_n | \boldsymbol{\mu}_j, \Sigma_j\right)} + \lambda.$$

Multiply by π_k and summing over k yields $N = -\lambda$. Thus

$$\pi_k = \frac{N_k}{N},$$

so that the mixing coefficients is given by the average responsibility that components carry in explaining the data points.

It is worth emphasising that the results obtained do not constitute a closed-form solution for the parameters of the mixture model, because the responsibilities of $\gamma\left(z_{nk}\right)$ depend on those parameters in a complex way, as stated in (2). Therefore, we first choose some initial values for the means, covariances and mixing coefficients. Then, we alternate between the two steps of the EM algorithm: step E (expectation) and step M (maximisation). During step E, we use the current values for the parameters to evaluate the posterior probabilities. We then use this probabilities, in

the M step, to re-estimate the means, covariances and mixing coefficients. Note that, in doing so, we are firstly evaluating the means, then the covariances and finally the mixing coefficients. This algorithm tends to require many more iterations compared to the K-Means algorithm to converge, also requiring more computations at every cycle. It is therefore common to run the K-Means algorithm first in order to find a suitable initialisation for a Gaussian mixture model. The means are the centroids of the clusters, the covariances are set as the covariances of the clusters and the mixing coefficients are set to be the proportions of points belonging to the cluster.

References

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer New York Inc., New York, NY, USA, 2006.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Data Mining and Knowledge Discovery*, pages 226–231. AAAI Press, 1996.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[Hun07] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[JW07] Richard A. Johnson and Dean W. Wichern. *Applied multivariate statistical analysis*. Pearson Prentice Hall, Upper Saddle River, NJ, sixth edition, 2007.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Pr., 2014.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[vdWCV11] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[ZWM14] Mohammed J. Zaki and Jr. Wagner Meira. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, Cambridge, UK, May 2014.