Aprendizagem Automática

Assignment 2 - Clustering

Andrea Mascaretti N52222 Daniel Pimenta N45404

7th 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

1.2 Assessing the fitness of clustering

1.3 The USGS Catalog

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]

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

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

[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.

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).

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

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

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

   (c) Set $\mu := \mu + 1$

3. Termination occurs when cluster assignments stop changing

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

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:

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

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 nth 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 kth 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. 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.

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

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