Machine Learning – Report 2

David Mendes (44934) & Francisco Cunha (45412)

December 9th, 2017

1 Introduction

For this project, we were given a data set with all the seismic events of magnitude at least 6.5 in the last 100 years, obtained from the USGS catalog, with the goal of examining the performance of three different clustering algorithms (K-means, Gaussian Mixture Models and DBSCAN) in this problem of clustering seismic events.

To do so, we varied these algorithms' main parameter values and analyzed the corresponding behaviors, in the context of our problem (with the help of various evaluation indexes, both internal and external). For the particular case of DBSCAN, this involved some research on the original paper of the algorithm. To finish off, we picked the algorithm that we considered to be the most effective in this context and documented the entirety of our approach.

2 On Shuffling and Pre-Processing

a. Shuffling

Data shuffling is a crucial step when we intend to split our data set, as we want to avoid any bias or patterns formed by chance in the split data sets before training our models. This was the case in our previous project, where we did split our data into subsets. However, for this second project, we were faced with an unsupervised clustering problem, and so there was no need to split our data. Thus, absolutely no need for shuffling.

b. Standardization

Standardization was a more sensitive topic, considering this decision (of whether or not we should standardize our attributes) is highly dependent on the data itself.

Standardizing is a *must* when our attributes are not comparable (*i.e.* are of incomparable units) because it will attempt to give all variables an equal weight. The thing is, even though our attributes, latitude, and longitude, were in the same units, they didn't even have an equal variance; whilst longitude ranges from *[-180, 0, 180]*, latitude ranges from *[-90, 0, 90]*.

Yet, they had a particularity: a **strong, well-defined meaning** in the context of the problem. Bringing them both onto the same range would distort their actual meaning - this is, once again, because of the specific nature of our problem, where we're working with geographic coordinates - and so we decided not to standardize it.

3 K-Means

a. The Algorithm

For this project, we used *scikit-learn's KMeans*, an implementation of Lloyd's algorithm for K-means. It is a prototype-based clustering algorithm that attempts to cluster data by trying to separate it into **n** groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares (WCSS).

In order to achieve this, only a simple distance metric is involved in the measurement, typically the Euclidean distance. This means that our data set is partitioned into a Voronoi tessellation, and so the algorithm will not be able to find concave clusters (only convex).

It will also perform poorly when confronted with anisotropic-shaped (*i.e.* not uniform in all orientations) or non-globular clusters. In fact, because it relies only on centroids to represent the clusters, it is not sensible to covariance nor variances, and so it tends to find clusters of comparable shape and spatial extent - usually sphere-shaped, when using the Euclidean distance. This is one of the main problems of prototype-based clustering (well, not *necessarily* a problem, but more of a *limitation* in terms of use cases).

Besides being prototype-based, K-means is also a complete clustering algorithm (with no explicit notion of noise), and so every single point of the data set will be assigned a cluster. Furthermore, it is an exclusive clustering algorithm, meaning it uses hard partitioning to assign each object to *exactly* one cluster. Once again, these characteristics may or may not be problematic; it's highly dependent on the problem's domain and on the data set we have.

b. In the Context of our Problem

For this problem, we think that not being sensitive to noise is a clear disadvantage, for obvious reasons. Furthermore, the sphere-shaped clusters that the algorithm famously produces don't seem particularly well-fitted for our data set, as many of the "original" groupings appear to be non-globular and overall anisotropic.

4 Gaussian Mixture Models (GMM)

a. The Algorithm

As we've seen before, there is no "maybe" in K-means; a point either belongs to a cluster or it does not. However, for most real cases, we will be faced with situations where two (or more) clusters meet. Let's pick the example of the two meeting clusters, say A and B. Using K-means' hard partitioning approach, we would get a binary verdict - a point either belongs to cluster A or to cluster B. The thing is, because we're on a "border" situation here, this clustering seems a bit weak. Instead, we would like to be able to say that there is some chance that our point belongs to cluster A, but there is also some chance that it belongs to cluster B. K-means clearly cannot give us such an output. GMM, however, can.

Gaussian Mixture Modelling takes an interesting approach: it's a clustering algorithm where clusters are defined by probability distributions. It's arguably the most popular variant of Expectation-Maximization (EM), where it is used a probabilistic model that assumes that all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters, namely mean and covariance.

Notice how this is the biggest difference between GMM and K-means. GMM is a variant of EM, and K-means is usually considered to be a special case of EM (where the covariance is simply not taken into account), and thus they have a lot in common.

However, because GMM takes variance into consideration, it uses *soft* assignments (and not hard assignments!) to estimate the most likely class for each observation. Just like K-means, GMM will not be able to find concave clusters. Yet, this sensibility to variance and covariance effectively makes it more flexible than K-means, in terms of cluster shape.

b. In the Context of our Problem

Same as the K-means algorithm, the Gaussian Mixture Models clustering technique suffers from noise insensitivity. However, we expect it to perform at least slightly better than the former, as soft partitioning will stay away from the abrupt spherical-shaping and move towards more stretched, elliptical clusters.

5 Density-Based Clustering (DBSCAN)

a. The Algorithm

Density-Based Clustering is another approach that intends to solve many of the problems of prototype-based clustering (namely its performance when faced with non-globular clusters, or clusters with different variances). Before diving into how the algorithm works, we will, for completeness purposes, briefly display some of the core concepts behind DBSCAN.

- Let *eps* be the maximum distance between two samples for them to be considered as in the same neighborhood.
- Let *minPts* be the number of samples in a neighborhood for a point to be considered as a core point (this includes the point itself).
- Let *N.eps* be the neighborhood of each point as the set of points within distance *eps*.

A point **p** is a core point if and only if the number of points in the *N.eps* of **p** is at least equal to the value of parameter *minPts*. Furthermore, a point **q** is reachable from **p** if and only if **p** is a core point and **q** is in the neighborhood of **p**, or in the neighborhood of any core point that is reachable from **p**.

With this information in mind, we can proceed to how the algorithm *actually* works: For each point **p**, if the number of points in the neighborhood of **p** is less than *minPts* (*i.e.* p is not "surrounded" by enough points), **p** is presumed to be noise. Otherwise, a cluster is created for **p**, which is a core point, and all neighbors of **p** are added to that cluster. If any neighbor of **p** is also a core point belonging to another cluster, the clusters are merged.

b. In the Context of our Problem

The first (and arguably most obvious) difference between this algorithm and the former two is that DBSCAN does **not** require one to specify the number of clusters in the data *a priori*. Usually, this is a positive aspect, as coming up with a "great" or "optimal" number of clusters is a rather tricky task. The *eps* and *minPts* parameters can also be set "manually", but doing that generally requires a considerable amount of domain expertise and superb understanding of the data.

Another important aspect is that out of the three clustering algorithms we approached during the project, DBSCAN is the only one that is sensitive to noise. Whilst the former two will necessarily assign each point to some cluster, the DBSCAN algorithm will classify some points as noise (as explained above). You can easily assert that by running the Mollweide projections for each of the algorithms' results; only the DBSCAN plot will display (some of) the original noise points.

Finally, due to its nature, DBSCAN can find concave, non-linearly separable clusters. As mentioned earlier, this definitely isn't the case for K-means or GMM. Furthermore, we think that this characteristic will be advantageous in the context of our problem, as many of the fault lines do appear to form non-linearly separable clusters.

6 Scores and Evaluations

a. Why use Validation?

Validation is a great way to **a)** compare clustering algorithms and **b)** optimize and choose clustering parameters. To do these evaluations, we used both internal and external indexes. The difference between the two is objectively very simple: whilst internal indexes evaluate the quality of clustering without reference to external information, external indexes compare the clustering results obtained with the ground truth - externally known results.

b. Evaluation Based on Internal Indexes

Two of the most basic internal index evaluation properties are cohesion and separation. Cohesion is a measure of the similarity of points within each cluster. In other words, it measures how closely-related are objects in a cluster. Separation, on the other hand, measures how distinct/well-separated a cluster is from other clusters.

We didn't explicitly use these in the project, but we used the Silhouette score, which combines ideas from the former two. This silhouette shows which objects lie well within their cluster, and which ones are merely somewhere in between clusters.

Silhouette coefficients (as these values are referred to as) belong to the *[-1; 1]* interval. Near (positive) one indicates that the sample is far away from the neighboring clusters. A value of zero indicates that the sample is on, or very close to the decision boundary between two neighboring clusters, and negative values indicate that those samples might have been assigned to the wrong cluster. Measuring the individual silhouette score for all points, we are effectively measuring how appropriately the data has been clustered, without reference to external information.

c. Evaluation Based on External Indexes

Throughout this project, the external validation indexes we used were the Rand Index, the Adjusted Rand Index (ARI), Precision, Recall and F1. We make use of an external index such as the Rand Index when we want to compare a clustering with some other partition of our data, such as another clustering or classification labels.

The adjusted Rand index is the corrected-for-chance version of the Rand index. Though the Rand Index may only yield a value between zero and (positive) one, the adjusted Rand index can yield negative values. Still, a negative ARI value would point towards actual patterns on the differences, not just randomness. From our research, this is quite rare and much more likely to arise from implementation errors than by anything else.

Finally, we didn't give much attention to the Precision and Recall indexes, as the F1 score already does the harmonic average between the two. In fact, they were mostly intermediate values.

7 Results Comparison and Analysis

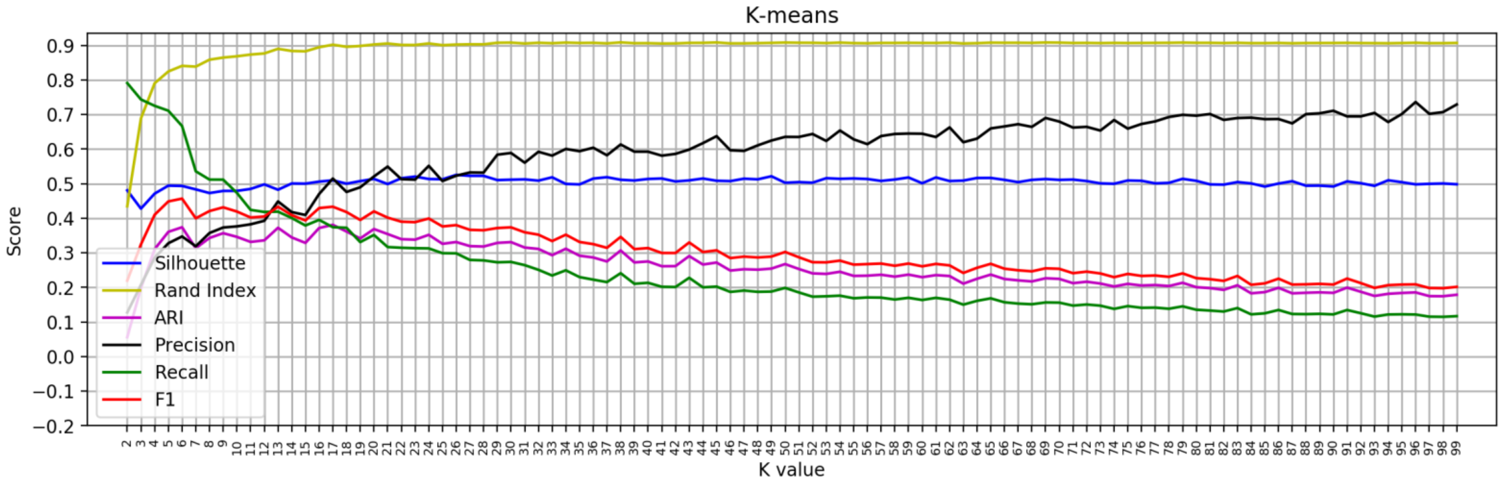
For each algorithm, we focused mostly on ARI, Rand Index, and F1 as the external index metrics, and the silhouette score to analyze the internal index factors. We excluded the precision and recall scores on this analysis, as we believe it would create a bias towards this scoring methods because they are integrated and figure in the F1 score, as it is the harmonic average of recall and precision.

Programmatically, we determined, for each one, the best parameter. This would be the one that achieves the best score sum of the four scores described earlier. We then compared this results with the empirical analysis of the graph. The silhouette score is ignored in the analysis of DBSCAN because of its negligence of the noise notion, and so the programmatic value determination will ignore this score in that particular case.

When it came to the range of the parameters to be tested, it came down to the meaning of each parameter. For K-means and GMM we decided on a range for k and number of components of 2 to 100 with a granularity of 1. This makes sense as each represents the number of clusters formed in each algorithm and our dataset has 98 different fault lines. So, we shot for a minimum of 2, one would not make any sense, and a little above the number of faults, without overshooting it. Apart from the graphs shown in the next few pages, we tested with a bigger range, meaning more than 100 in the upper bound of the range, but the results did not change, so we stuck with the above range to improve the legibility of said graphs.

For the Eps parameter in DBSCAN, we looked into the “elbow” method that will later be described and chose a range that would make sense looking into the slope of the k-dist graph. Meaning, an upper bound where the slope was still quite aggressively changing and a lower bound where said slope registered almost no changes. With that in mind, we settled for a range of 60 to 800, with a granularity of 10.

a. K-Means



By analyzing this graph, we can see that the relevant scores, tend to increase in an early phase, as the number of clusters increases, as expected. The key here is to determine *when* the increasing of the number of clusters starts to negatively impact the overall score. Programmatically and as described earlier, we perceived the best parameter to be 20 clusters. Confronting this result with the empirical analysis, we can see that there are at least two points in which the value should also be considered, 6 and 13.

To decide which one we would choose as the parameter value, we plotted each one in the globe projection as follows:

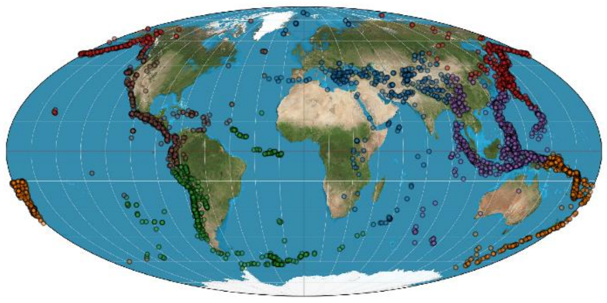


Figure 1 - Plot with k = 6

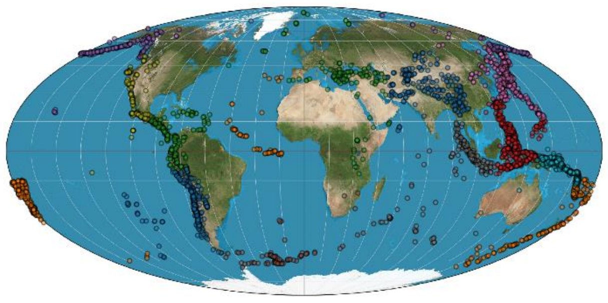


Figure 2 - Plot with k = 13

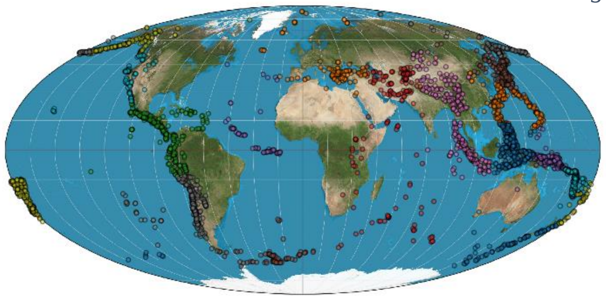
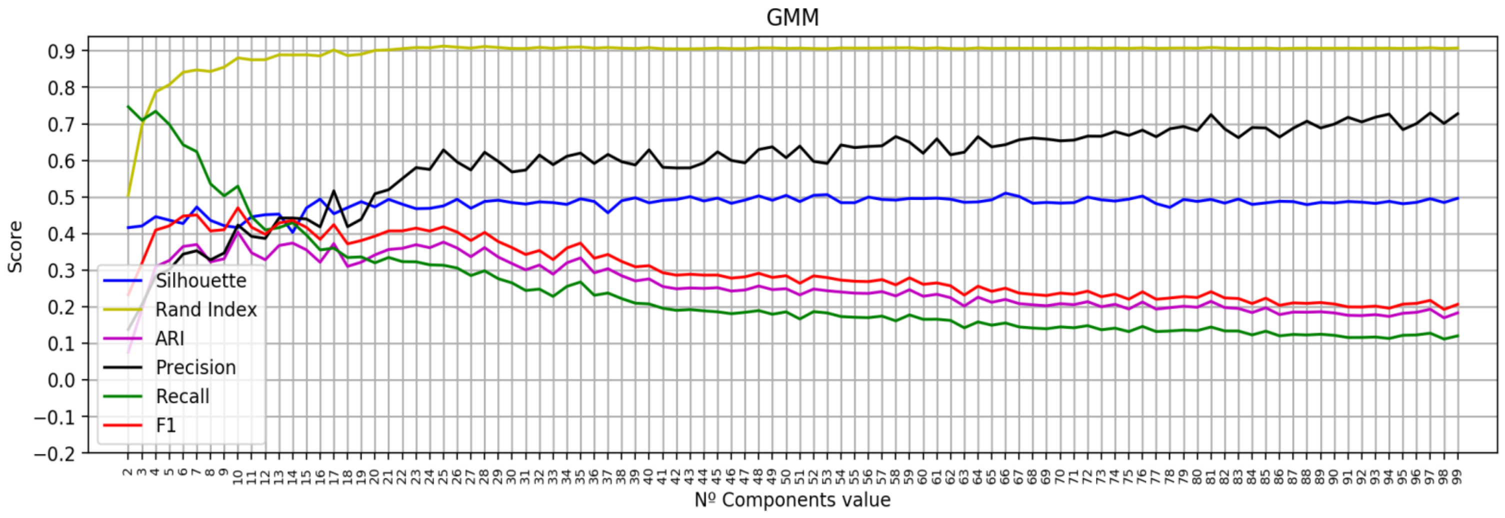


Figure 3 - Plot with k = 20

As we can observe, the one that better portrays the original projection is the one with 20 clusters, not just because it has more clusters - which goes in line with the expected result - but also because the clustering's separation is closer to the expected outcome.

b. Gaussian Mixture Modelling



In the GMM graph, and similarly to what we saw with the previous algorithm, the scores also tend to increase in an early phase (apart from recall). Programmatically, the number of components we obtained was 16. Comparing this result with the empirical analysis, we can see that points 8 and 11 should also be taken into account, as they have peaks within scores in their respective intervals. Even though worth analyzing, with 8 components the scores have small peaks within its closest neighbors. Still, comparing it with the 16 components scores, we can see a considerable increase in the Rand-Index score. Given that fact we discarded the value of 11 and plotted the projection for 8 and 16 components for comparison:

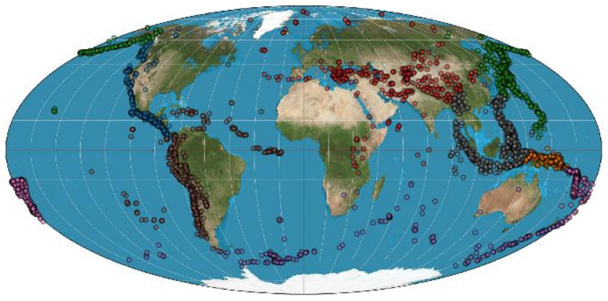


Figure 4 - Plot with 8 Components

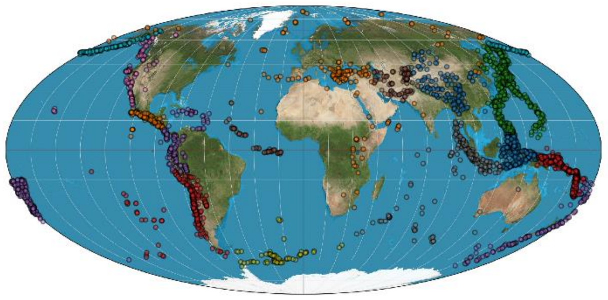
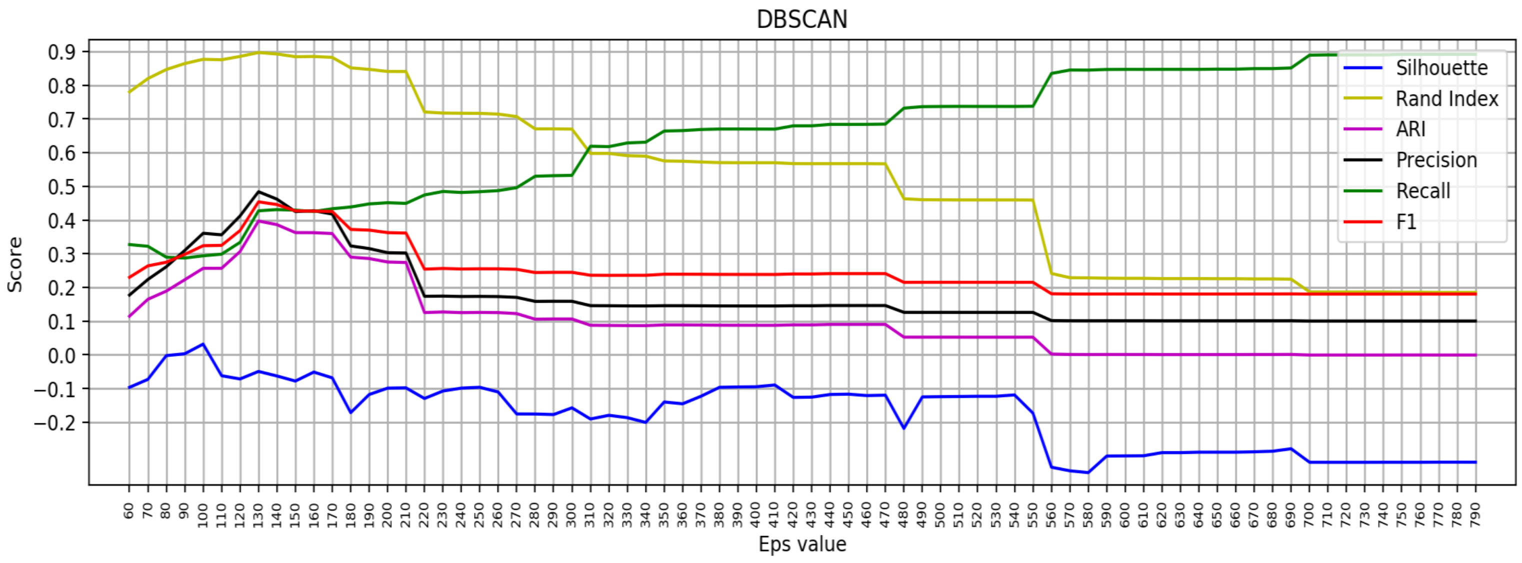


Figure 5 - Plot with 16 Components

We can see that, with 16 components, we are able to distinguish faults on the Americas and Asia Region that were bundled together in the 8 Components version, for that reason we choose the value 16 as the best for our number of components in this sample with the Gaussian Mixture Model.

c. DBSCAN



The DBSCAN graph analysis is the easiest regarding the relevant scores (Rand Index, F1, and ARI). We exclude the silhouette score from this analysis, even though it is displayed, due to the fact that the silhouette score has no notion of noise, and so it would negatively impact the overall score, in an unfair way. This was mentioned earlier in the report.

Programmatically we obtained an *eps* value of 130, and we have no way to contradict this result since the scores within the graph peak exactly at that value, apart from the recall and silhouette scores that we, as previously stated, discarded in this analysis.

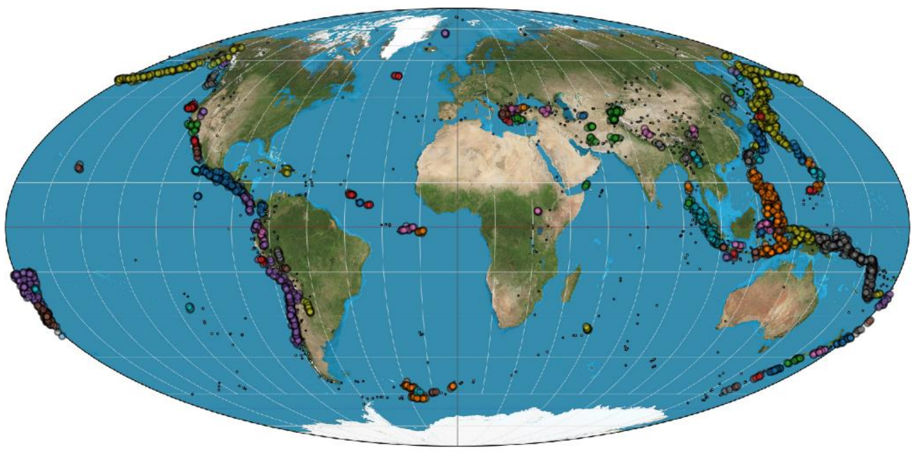


Figure 6 - Plot with Eps Value of 130

8 Implementation Details on our Method to Choose *eps*

Following the analysis of the recommended paper, we followed the described method for determining the *eps* parameter considering a Minimum of Points (*minPts*) of four.

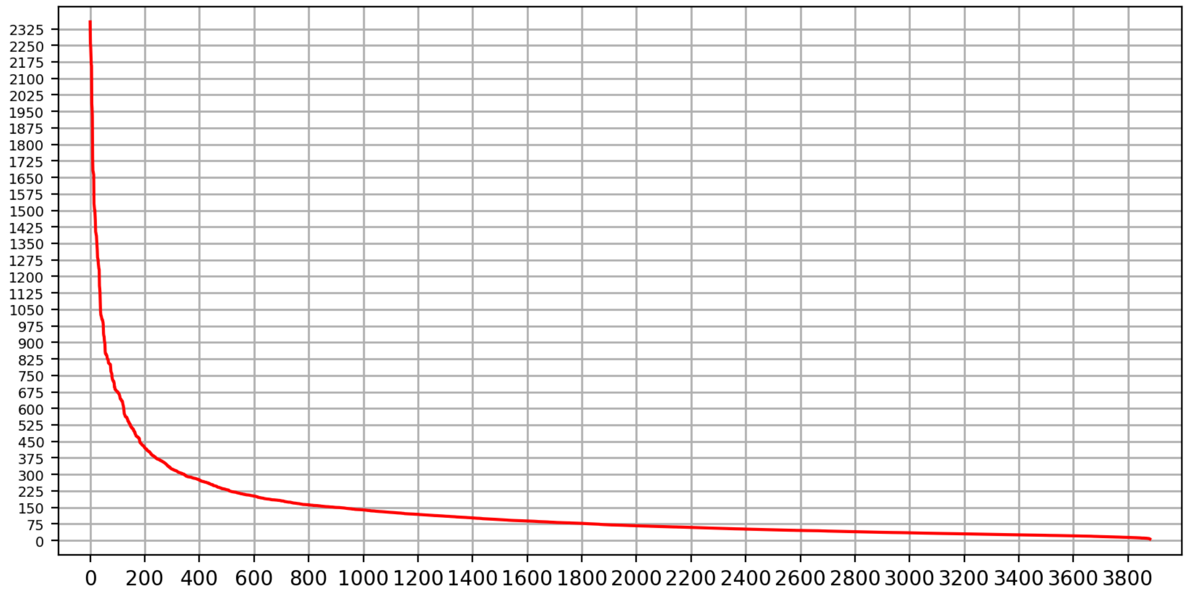
We used the k-nearest neighbor's algorithm to be able to determine the distance of each point to its 4th closest neighbor. With that, we were able to plot the graph of the sorted distances and be able to determine the pretended parameter.

Considering that, for each point in this graph, if we use its distance to the 4th closest neighbor to establish the *eps* parameter, every point with a larger distance to its 4th neighbor will be considered as noise. This is because the algorithm will not be able to include it in a cluster, as the distance setting will make it, so the points will no longer be reachable and thus not be included in a cluster.

On the other hand, all points with a smaller distance to their 4th closest neighbors are *guaranteed* to belong to some cluster. The challenge here is to select an *eps* value such that we maximize the k-dist value in the "thinnest" cluster of the sample, which there is not really a fool-proof way to accomplish.

To select the threshold point, we followed this mindset: we should choose a point so that choosing it over it's the closest point would not bring *much* difference. So, we ended up choosing a point in the elbow of the graph such that the slope of the graph would not change, in our eyes, in a meaningful way.

Considering the obtained graph shown below:



By utilizing the preferred methodology, we were able to pinpoint, the *eps* value to 160.

Given that parameter, we plotted the results with the provided globe plotting method, which yielded the following result:

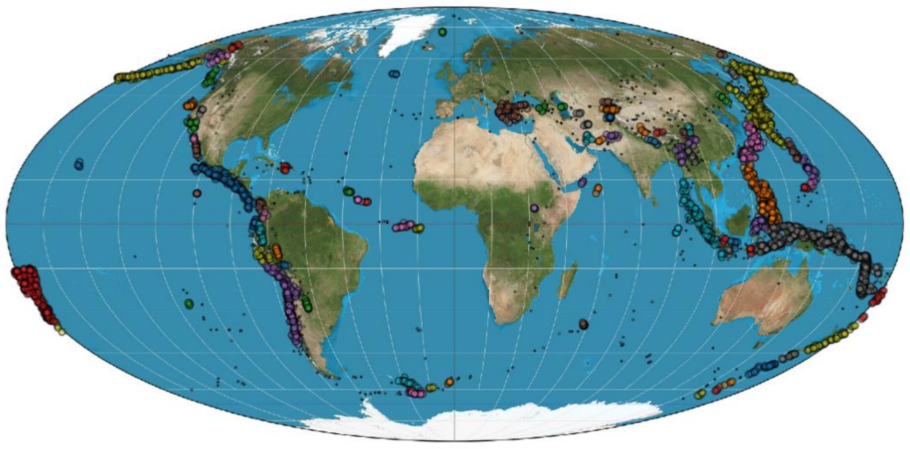


Figure 7 - Plot with Eps Value of 160

In comparison to the original image, we can see that the algorithm clusters some points, that were meant to be noise, and mistakenly splits some of the original clusters into smaller ones, in cases where they are geographically really close to each other. Albeit, it still maintains most of the original projection characteristics, but mostly with a higher level of granularity.

9 The Best Algorithm: Conclusion and Analysis

Considering the previous plot projections for each algorithm, we empirically determined the best algorithm for our data set.

We already knew that K-Means and GMM would start at a disadvantage, at least, in comparing the results with the original faults representation, due to the fact, that they have no notion of outliers, meaning that every point would belong to a cluster no matter what. DBSCAN, on the other hand, considers that points that are isolated and too far from any other point are assigned to a special cluster of outliers, and thus it would give it a clear advantage as the external indexes scoring goes.

Another advantage of the DBSCAN is the fact that since the clusters are formed without assuming any shape, they are created by looking for neighborhood points. Thus, clusters of various shapes can be detected, which makes it so that in most occasions DBSCAN is a good candidate for clustering geolocated events.

With all that in mind, it was with no surprise, that we realized that DBSCAN was the best algorithm in the context of our problem, not only due to its outlier detection but also on it's improved model granularity on comparison with the other models. Albeit we can't discard the results obtained from the other algorithms - as unsupervised learning thrives on searching unknown knowledge - the clustering provided by the algorithms can still be meaningful in another interpretation, but just not in this particular fault line association, due to its characteristics.

10 Bibliography

- ⇨ http://iopscience.iop.org/article/10.1088/1755-1315/31/1/012012/pdf
- ⇨ https://en.wikipedia.org/wiki/DBSCAN
- ⇨ http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
- ⇨ http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
- ⇨ https://en.wikipedia.org/wiki/F1_score
- ⇨ https://www.youtube.com/watch?v=REypj2sy_5U
- ⇨ http://eecs.oregonstate.edu/research/multiclust/Evaluation-4.pdf
- ⇨ http://aa.ssdi.di.fct.unl.pt/files/LectureNotes.pdf