UNSUPERVISED LEARNING - MIDTERM ASSIGNMENT

Guy Van-Dam

ABSTRACT

Clustering 3 different data-sets using 5 different **Clustering Algorithms**, showing results and analyzing performances for each, in an attempt to find the best algorithm for clustering a given data-set.

Finding the *K-Means* algorithm to be the best preforming algorithm in terms of internal and external measures for the given data-sets.

All source code can be found in the GitHub repository

UnsupervisedLearningBIU

1 INTRODUCTION

When building a model, we need to experiment with different Machine-Learning methods, as different methods are built for different tasks. When we do not have a lot of information about our data, this step is crucial if not necessary for building an effective and efficient model. Trying different algorithms and analyzing and comparing their performances against each other will guide us towards building the right model for the task at hand.

When is comes to Unsupervised Machine-Learning, where we rarely have external labels, and we do not have a lot, if any information about our data. And so, we have to try different clustering algorithms and compare their results for classifying our given data-set in the most accurate way

2 METHODS

2.1 RANDOM SEED

Many of the functions that were used were randomly initialized, so a fixed random state was required for reproducibility. I used 42 as a fixed random seed.

2.2 PREPARING THE DATA FOR CLUSTERING

The data-sets were given to us *CSV* files, using the *Pandas* library we can read and store them as *Pandas' DataFrame* objects. Due to memory and run-time constraints, data-sets 2 and 3 needed to be reduced to 14,000 points, while data-set 1 could stay whole at about 12,000 points. The *DataFrames* were sampled using the built-in *sample* method, with our fixed random seed of 42.

When looking at the data we can clearly see that some of the components are not relevant, they do not give us information about the data, or they were asked to be deleted. For this case we can just delete the *DataFrame* column associated with that component.

In data-set 1 a lot of the components were a mix of numbers and strings with no clear order between them. To combat this issue, we can replace each value with a number, starting from 1 and use a *Python Dictionary* to store the original values. We now have a *Pandas DataFrame* with 12,000-14,000 numeric points and no irrelevant information.

Each data-set were given to us with components to use as external classifiers, so this is the point we need to save our the columns representing those components in a separate *DataFrame*

2.2.1 DIMENSION REDUCTION - PRINCIPAL COMPONENT ANALYSIS

Although we eliminated some columns, our data-sets were all above 3 dimensions, so a dimension reduction algorithm needed to be used for visualizing the results. The dimension reduction algorithm of choice was **Principal Component Analysis**, or **PCA** with a the fixed random seed mentioned above.

After normalizing the data using functions from the *Scikit* library, it is passed through the PCA algorithm, reducing the data to around 10% of the original size, as the average number of dimensions of one point is 20 on average.

We normalize and reduce the data dimensions using the *reduceDimensions()* method of the *DimReductionAlgorithm* object in *DimReductionAlgorithm.py*

After sampling, deleting, and normalizing. the data is *prepared* for clustering.

This whole process is done by calling the method *prepareDataset()* found in the *DataSet* object in *DataSet.py*

2.3 Clustering Algorithms

2.3.1 K-Means

A very popular and intuitive hard clustering algorithm. After randomly initializing k centeroids $\{c_1, ..., c_k\}$ we repeat the following 2 steps:

Assignment Step where are each point is assigned to a centeroid with the minimum Euclidean distance, $C_i = \left\{x_p : \left\|x_p - c_i\right\|^2 \leq \left\|x_p - c_j\right\|^2 \forall j : 1 \leq j \leq k\right\}$

Update Step recalculate the centeroids as the mean of points in C_i

We stop when there is no change in the *Update Step* or we reached the maximum number of iterations.

2.3.2 Gaussian Mixture Models - GMM

This is a soft clustering, *EM* algorithm, It assumes our data is comprised of n **multi dimensional Gaussian** distributions. $\left\{C_i\right\}$. As with *K-Means* it repeats 2 steps

Expectation Step we calculate $\gamma_{i,k}$ - the probability the x_i is generated by Gaussian Distribution C_k

Maximization Step we recalculate, the mean μ_k, variance Σ_k and weight ϕ_k

Similar to *K-Means* we repeat these 2 steps until there is no change in the *Maximization Step*, or we got to the maximum number of iterations. I used the GMM algorithm with a separate co-variance matrix Σ_k for each C_i

2.3.3 Fuzzy-C-Means

This algorithm is very similar to K-Means as it try to minimize the same objective function. Unlike *K-Means* it dose not assign a point to a cluster, rather assign a weight $w_{i,j}$, the probability of x_i to be in C_j and follows the same steps with the euclidean distance multiplied by the corresponding weight as a coefficient.

2.3.4 Agglomerative

A type of *Hierarchical* Clustering. with each point starting as its own cluster, and pairs of clusters are merged in a *greedy* manner that minimally increases a given linkage distance. I used was the *Ward* linkage criterion which minimizes the **variance of the clusters being merged**

2.3.5 Spectral

Using the *affinity matrix* A and *Degree matrix* D of the graph representation of our data, we calculate the **eigenvectors** and **eigenvalues** of the *Laplacian* matrix $L = D - A$.

From here the spectral embedding can be bi-partitioned, or separated by a clustering algorithm like *K-Means*.

I used an affinity matrix constructed using the *radial basis kernel function* and partitioning with *K-Means*.

2.4 Visualizing Clustering Results

After the data is prepared, we use the clustering algorithms classes from *Sci-Kit* and the *fit_predict* method within them. This method fits the algorithm to the data and returns the clustering labels, red-hot representation of the cluster. Using the output labels, we can easily plot the clustering, using the *scatter* method from *Matplotlib*. We plot the first and second component of the data and use the label as a color.

Running *PlotClusters.py*, will save a figure of the clustering in *.png* file

2.5 Fitment To External Classification

2.5.1 Kullback–Leibler Divergence

$$D_{\mathrm{KL}}(P\|Q) = \sum P(x)\log\left(\frac{P(x)}{Q(x)}\right)$$

The K-L divergence is a measure of how one probability distribution is different from a second. Using this measure we can calculate the performance of prediction labels of the clustering algorithm against the external labels given to us, as we want to get as little differences between the distributions of the predictions and ground truth.

With the columns we saved before, we calculate the number of different values in each column and cluster the data with that number of clusters. Then, we calculate the $D_{\mathrm{KL}}(P\|Q)$ where P is the probabilities of the external classifiers and Q is the probabilities of the labels we get from each clustering algorithm. this step is repeated with 30 different random states for the clustering algorithms, we save the results in a *CSV* file in the *ExteranlLabels* folder under *Results*.

By running *FitExternalClass.py* the *CSV* file is created and saved.

2.6 Finding The Optimal Number Of Clusters

To find the optimal number of clusters for each clustering algorithm, we cluster the data with different number of clusters, evaluate the algorithm performance and choose the number of cluster which performed best. The measure of performance I used was the ***Silhouette Score***. The number of clusters ranged between 2 and 10.

2.6.1 Silhouette Score

The *Silhouette Score* is an internal measure of performance of a clustering algorithm. We start by defining

a_i to be the mean distance of point x_i from the other points in **its** cluster and

b_i to be min$\{$mean distance of point x_i from the points in the **other** clusters$\}$.

We can now define the *Silhouette Score* for a point x_i as $s_i = \frac{b(i)-a(i)}{\max\{a(i),b(i)\}}$.

We calculate the *Silhouette Score* for each n, where n is the number of clusters and take the n with the largest *Silhouette Score*. In order to avoid chance and get a more robust result, we repeat this process for 10 different random states. For each random state, we save a figure of the silhouette score for n for all the algorithms, And the scores value in a *CSV* file in the *OptimalNClusters* folder under *Results* As we will see in section **3**, a statistical test would have failed as the chosen n was identical for all of the random states.

By running *OptimalNClusters.py* the *CSV* file and figures are saved under the corresponding folders.

2.7 Comparing Performances - Statistical Test

Because all the clustering algorithms I have used are randomly initialized, different random states can affect its performance. So, when defining a measure of performance, we most perform a statistical test to determine if our sample of random states accurately represents the algorithm true performance. Knowing what the optimal number of clusters for each algorithm is, we can compare the algorithms performance against one another.

2.7.1 One-Tailed Two-Sample T-Test

The *one-tailed two-sample t-test* is a statistical test for difference between the 2 samples means. We can use this test for testing if a mean of the first sample is greater than the mean of the second sample. To get the *p-value* for the *Null Hypothesis* $H_0 : \mathbb{E}[b] \geq \mathbb{E}[a]$ we take the we need to manipulate the output *p-value* from the two-sided test as

$$\textit{p-value} = \begin{cases} 1 - \frac{p_{output}}{2}, & \text{if } t - statistic_{output} < 0 \\ \frac{p_{output}}{2}, & \text{otherwise} \end{cases}$$

If we get a *p-value* lower than 0.05, we can reject the null hypothesis, and say that $\mathbb{E}[a] > \mathbb{E}[b]$

For each clustering algorithm, we calculate the *Silhouette Score* for 30 different random states, which gives us a list of scores. We can now test for mean differences with the statistical test seen above.

We can now determine that $algorithm_a$ to be better performing than $algorithm_b$ with their respective optimal number of clusters for each data-set, by evaluating their true mean *Silhouette Score* From here we can easily find the best performing algorithm, by searching for the maximum with our newly defined order.

We get our test results by running *StatisticalTest.py* which creates a *CSV* file with the relevant information

3 Results

3.1 Data-Sets

3.1.1 Data-Set 1 - Online Shoppers Purchasing Intention

This data-set consists of 12,330 points with 18 features. Each point would belong to a different user in a 1-year period to avoid any tendency to a specific campaign, special day, user profile, or period.

3.1.2 Data-Set 2 - Diabetes 130-US Hospitals For Years 1999-2008

Represents 10 years of clinical care at 130 US hospitals. It includes 101766 points with more than 50 features of numerical and literal data.

3.1.3 Data-Set 3 - Clickstream Data For Online Shopping

It Contains information on clickstream from online store offering clothing for pregnant women. Data are from five months of 2008 and include, among others, product category, location of the photo on the page, country of origin and product price in US dollars. It has 165474 points with 14 features. With all but one numerical features.

3.2 Visualizing Clustering Results

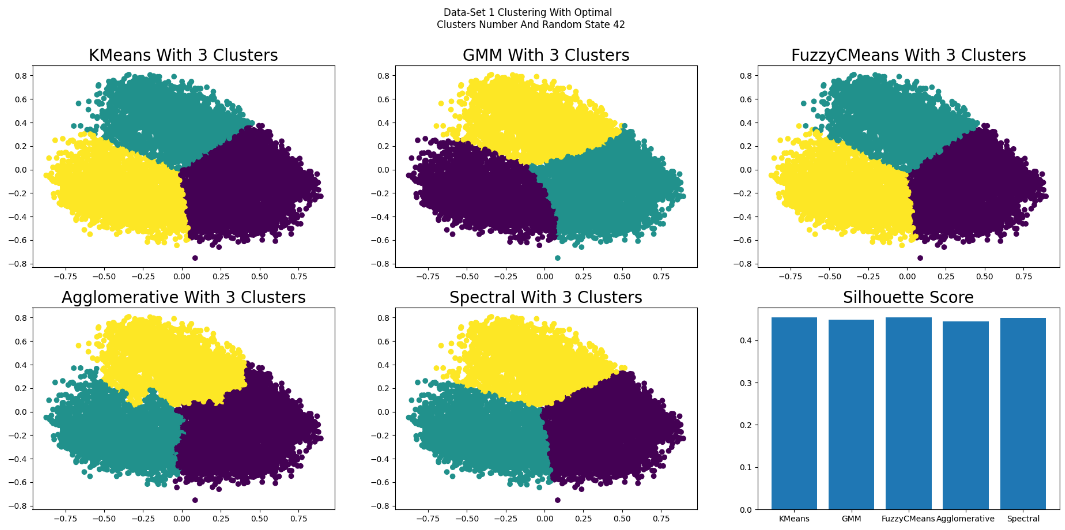

Figure 1: Data-Set 1 Clustering

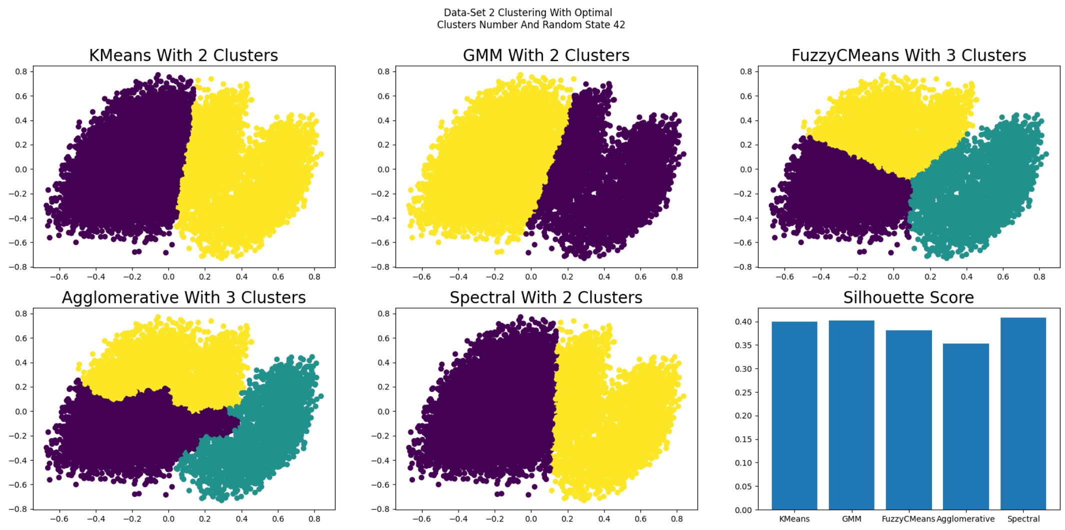

Figure 2: Data-Set 2 Clustering

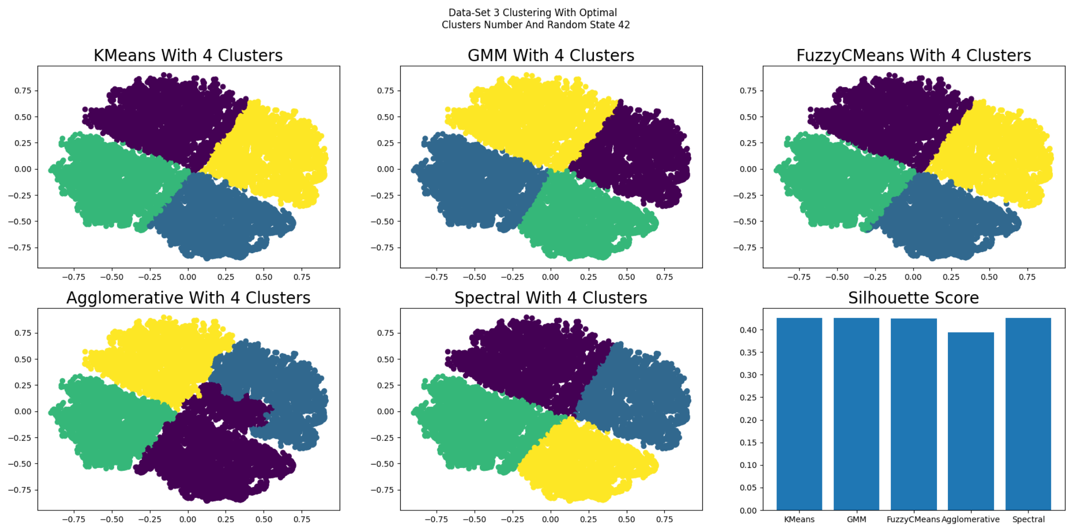

Figure 3: Data-Set 3 Clustering

3.3 Fitment To External Classification

3.3.1 Data-Set 1

Table 1: Average Of K-L Divergence Across 30 Random States Data-Set 1

Classifier	K-Means	GMM	Fuzzy-C-Means	Agglomerative	Spectral
Revenue	0.085	0.088	0.093	0.160	0.090
Visitor Type	0.092	0.084	0.087	0.086	0.088
Weekend	0.079	0.081	0.083	0.135	0.082

3.3.2 Data-Set 2

Table 2: Average Of K-L Divergence Across 30 Random States Data Set 2

Classifier	K-Means	GMM	Fuzzy-C-Means	Agglomerative	Spectral
Gender	0.052	0.074	0.033	0.122	0.064
Race	0.136	0.173	0.133	0.134	0.258

3.3.3 Data-Set 3

Table 3: Average Of K-L Divergence Across 30 Random States Data Set 3

Classifier	K-Means	GMM	Fuzzy-C-Means	Agglomerative	Spectral
Gender	0.177	0.194	0.170	0.220	0.305

3.4 Finding The Optimal Number Of Clusters

3.4.1 Data-Set 1

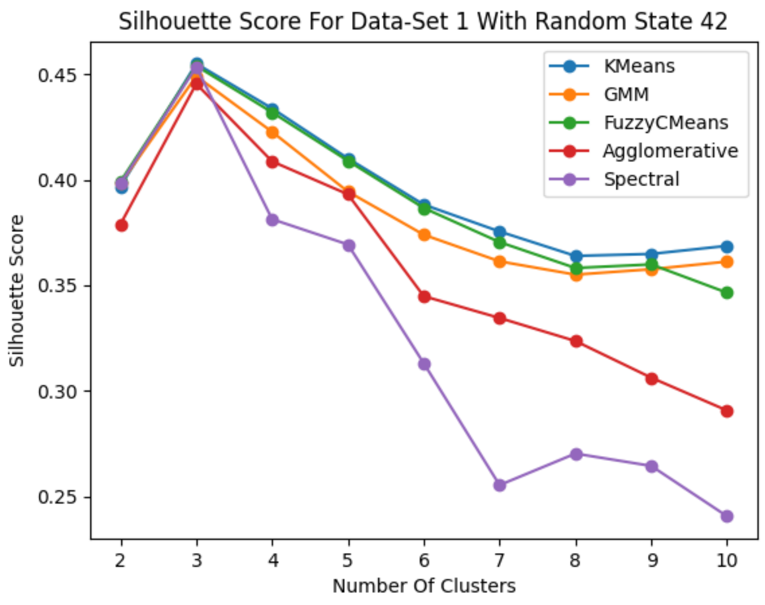

Figure 4: Silhouette Score For Data-Set 1

3.4.2 Data-Set 2

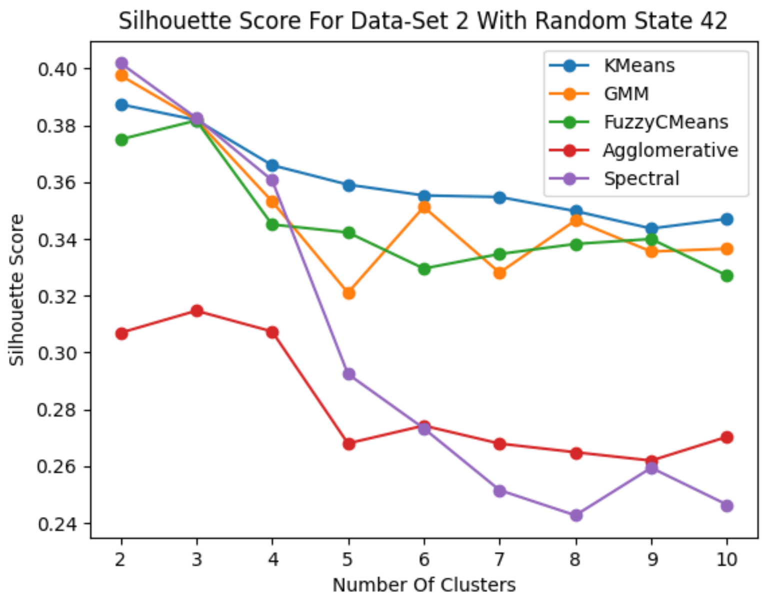

Figure 5: Silhouette Score For Data-Set 2

3.4.3 Data-Set 3

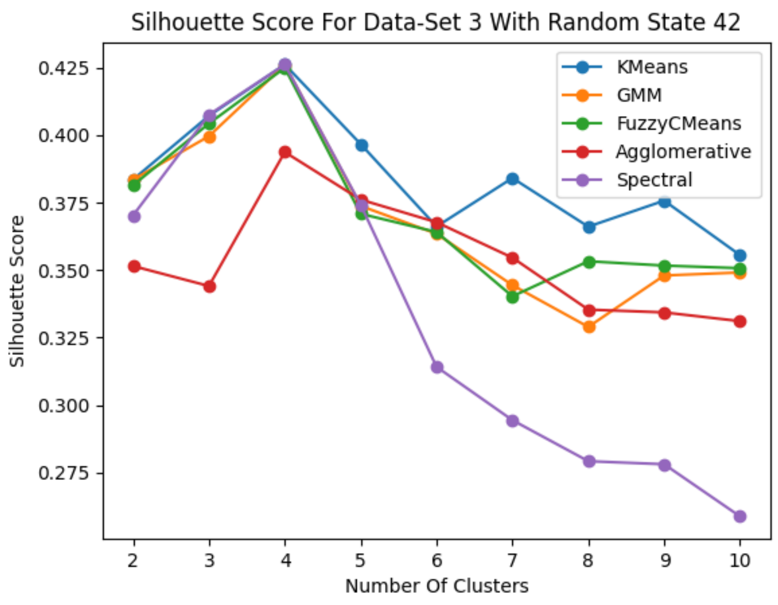

Figure 6: Silhouette Score For Data-Set 3

In the *CSV* files in the *OptimalNClusters* folder for each data-set, under *Results* we can see that the 10 selected different random states did not had effect the final result, the figures look very similar and the optimal number of clusters for every algorithm was the same across the different states.

3.4.4 Summary

Table 4: Optimal Number Of Clusters

Data-Set	**K-Means**	**GMM**	**Fuzzy-C-Means**	**Agglomerative**	**Spectral**
Data-Set 1	3	3	3	3	3
Data-Set 2	2	2	3	3	2
Data-Set 3	4	4	4	4	4

3.5 Comparing Performances - Statistical Test

3.5.1 Data-Set 1

Table 5: Average Of *Silhouette Score* Across 30 Random States Data-Set 1

H_0	***p-value***	***T-Statistics***	**Greater Mean**
$E[KMeans] \geq E[GMM]$	1	-117.542	KMeans
$E[KMeans] \geq E[FuzzyCMeans]$	1	-205.403	KMeans
$E[KMeans] \geq E[Agglomerative]$	1	-1953.983	KMeans
$E[KMeans] \geq E[Spectral]$	1	-73.131	KMeans

3.5.2 DATA-SET 2

Table 6: Average Of *Silhouette Score* Across 30 Random States Data-Set 2

H_0	***p-value***	***T-Statistics***	**Greater Mean**
$E[KMeans] \geq E[GMM]$	7.280×10^{-33}	-117.542	GMM
$E[GMM] \geq E[FuzzyCMeans]$	1	-57.222	GMM
$E[GMM] \geq E[Agglomerative]$	1	-136.304	GMM
$E[GMM] \geq E[Spectral]$	1	-9.858	GMM

3.5.3 DATA-SET 3

Table 7: Average Of *Silhouette Score* Across 30 Random States Data-Set 3

H_0	***p-value***	***T-Statistics***	**Greater Mean**
$E[KMeans] \geq E[GMM]$	0.993	-2.651	KMeans
$E[KMeans] \geq E[FuzzyCMeans]$	1	-106.662	KMeans
$E[KMeans] \geq E[Agglomerative]$	1	-3024.014	KMeans
$E[KMeans] \geq E[Spectral]$	2.60×10^{-5}	4.708	Spectral

3.6 SUMMARY

Table 8: Performance Summary

Performance	**Data-Set 1**	**Data-Set 2**	**Data-Set 2**
Best External Classifier Fitment	KMeans	FuzzyCMeans	FuzzyCMeans
Best Average Silhouette Score	KMeans	GMM	Spectral

4 DISCUSSION

We do no get a concise answer to what the best algorithm is for data-sets 2 and 3, as the algorithms that fitted best to the external classifiers did not had the best silhouette score in the statistical test. It might be due to the fact the we clustered data-set 1 whole, in contrast to data-set 2 and 3 which needed to be sampled intensively.

We can assume that ***K-Means*** was the best algorithm for **data-set 1** as it was the best performer in both the *Fitment To External Classifiers* and *Silhouette Score* tests.