SUPERVISED & UNSUPERVISED LEARNING - FINAL PROJECT

BY YUVAL BEKER, 209221621 & DAVID GURIEL, 211481130, BAR-ILAN UNIVERSITY

Abstract. **In this project we analyze the "credit-card frauds" dataset, using clustering methods and unsupervised learning.** We begin by examining the distributions of the given features of the data and the dependencies between them. Then we visualize the data statistics with graphs and histograms. Finally, we run five algorithms: K-means, Fuzzy C-means, GMM, Spectral Clustering & DBSCAN. We modify the feature vectors in different ways in order to optimize the performances, which are measured in terms of precision, recall and f-1 score. For each modification we run those algorithms and measure their performance.[1].

Part 1. - The Data

In general, the dataset represents information about a vast amount of credit-card transactions, performed on September 2013, during two whole days (i.e. 48 hours). However, for the sake of customers privacy, the data was not exposed to us. Instead of that, the creators performed PCA transformation on the data, and the result is 28 principle components.

In addition to the components, one can find the following columns:

- Time - number of seconds passed since the beginning of the measurement.
- Amount - the amount of performed transaction.
- Class - a binary value, determining whether the transaction is classified as a fraud or not.

There are quite many transactions - 284,807 totally, spread over 48 hours, which are 172,800 seconds (last transaction was made on second 172,792). Out of them, there are 284,315 non-fraud (negative) transactions, and only 492(!) positive ones. This results in a great amount of imbalance, and we tried to deal with it in our work.

Note: In all the algorithms, we attend to cluster the data into **2 clusters**, since the classification should be binary - is it a fraud or not.

Part 2. - Statistics

In this part, we review the data analysis we performed, in order to make insights about the data, and to determine whether to use the columns "Amount" and "Time" at the different algorithms.

Firstly, we calculated the mean and variance of the 30 features: the 28 we got from PCA, and "Time" and "Amount". We found that the mean of "Time" is 9481, the mean of each one of the 28 PCA features is 0 (due to the PCA algorithm that normalizes them all), and the mean of "Amount" is 8.834962.

As for the variances, they are all described on **table 0a**. One can observe that the PCA features variances are ordered in a decreasing order. We had plotted the features, and could see that the PCA features have a normal form. The rest of the features are shown as well on **table 0a**.

[1]the dataset is available here.

Feature	Variance
Time	$2 \cdot 10^9$
v1	3.836476
v2	2.72681
v3	2.299021
v4	2.004677
v5	1.905074
v6	1.77494
v7	1.530395
v8	1.426474
v9	1.206988
v10	1.18559
v11	1.041851
v12	0.9984
v13	0.990567
v14	0.918902
v15	0.8378
v16	0.767816
v17	0.721371
v18	0.702537
v19	0.66266
v20	0.594323
v21	0.539524
v22	0.526641
v23	0.389949
v24	0.366807
v25	0.27173
v26	0.232542
v27	0.162919
v28	0.108955
Amount	8.835
Table 0a	

	Time	Amount	Class
Time	1.000	-0.011	-0.012
v1	0.117	-0.228	-0.101
v2	-0.011	-0.531	0.091
v3	-0.42	-0.211	-0.193
v4	-0.105	0.099	0.133
v5	0.173	-0.386	-0.095
v6	-0.063	0.216	-0.044
v7	0.085	0.397	-0.187
v8	-0.037	-0.103	0.02
v9	-0.009	-0.044	-0.098
v10	0.031	-0.102	-0.217
v11	-0.248	0.000	0.155
v12	0.124	-0.01	-0.261
v13	-0.066	0.005	-0.005
v14	-0.099	0.034	-0.303
v15	-0.183	-0.003	-0.004
v16	0.012	-0.004	-0.197
v17	-0.073	0.007	-0.326
v18	0.09	0.036	-0.111
v19	0.029	-0.056	0.035
v20	-0.051	0.339	0.02
v21	0.045	0.106	0.04
v22	0.144	-0.648	0.001
v23	0.051	-0.113	-0.003
v24	-0.016	0.005	-0.007
v25	-0.233	-0.048	0.003
v26	-0.041	-0.003	0.004
v27	-0.005	0.029	0.018
v28	-0.009	0.01	0.01
Amount	-0.011	1.000	0.006
Table 0b			

Feature	Mean	Variance
Time	8075	$2 \cdot 10^9$
v1	-4.77	45.92
v2	3.62	18.38
v3	-7.03	50.46
v4	4.54	8.24
v5	-3.15	28.8
v6	-1.4	3.45
v7	-5.57	51.83
v8	0.57	46.12
v9	-2.58	6.24
v10	-5.68	23.94
v11	3.8	7.16
v12	-6.26	21.62
v13	-0.11	1.22
v14	-6.97	18.27
v15	-9.29	1.1
v16	-4.14	14.91
v17	-6.67	48.49
v18	-2.25	8.39
v19	0.68	2.37
v20	0.37	1.81
v21	0.71	14.94
v22	0.01	2.23
v23	-0.04	2.49
v24	-0.11	0.27
v25	0.04	0.63
v26	0.05	0.22
v27	0.17	1.89
v28	0.08	0.3
Amount	122.21	6575
Table 0c		

Next, we calculated the covariances and the correlations of every pair of features, this time including “Class”. As expected, we found that between each pair of different PCA features, the covariance is 0. As for the other features, we preferred to present the correlation values, instead of the covariances, in order to get a picture of the dependencies between them. We used the standard formula for correlation: $\rho = \frac{Cov(\mathbf{x},\mathbf{y})}{\sqrt{Var(\mathbf{x})}\cdot\sqrt{Var(\mathbf{y})}}$.

The correlations of “Time”, “Amount” and “Class” with respect to all the other features are listed in **table 0b**. The PCA features look independent, as the correlation coefficient between each pair of them is 0. From the table we see that no two features have an especially strong correlation, and in particular no PCA feature gives much indication of the class. The two features that have the largest correlation (in absolute value) with “Class” are “v14” and “v17”.

Later, we wanted to characterize the frauds data, and so we performed the same analysis just on the frauds (where “$Class$” $== 1$), as described on **table 0c**. The fraud (positive) data are less neatly organized. We can see

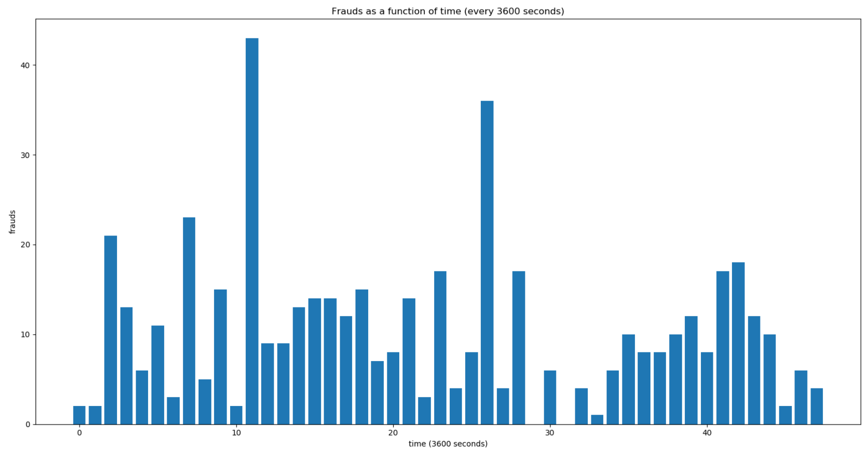

FIGURE 1. Histogram of the frauds over time (hours)

that some of the features have relatively large variances, and that they don't have a Gaussian form. The matrix of correlations is shown on **positive_correlations.xlsx**.

The last thing we'd like to present is histograms of the times and amounts of the frauds transaction (i.e. only the positive examples). We tried to plot the entire dataset, yet it turned out to be impossible, due to the massive amount of negative examples. In **figure 1**, one can see the distribution of the frauds over time, as we chose to run the function "get_hists_grouped" with a seconds parameter (called "step") value of 3600 seconds, i.e. an hour. This results in a distribution over hours (of whom we already know there are 48). A possible explanation for the low rate around the 30th hour, as well as the several first ones, could be the fact that it was night-time, so the criminals are less active.

On **figure 2**, we present the distribution of the frauds with respect to the amount of the transaction. Obviously, we did not show every transaction value, but rounded them to integers. Then, we filtered away the amounts that occured only 0 or 1 times. This left us with a reasonable diagram to show.

We can see that most of the frauds were actally of a very low rate - between a few cents to 2 dollars. This raises another issue regarding the imbalance of the data - **it is hard to identify large frauds**. Another justification of this argument is the low correlation between the "Amount" and "Class" features. On the other hand, one can claim that even finding the 1-dollar frauds is good, because the algorithms give us a classification method, that will also work on other data, and function as an additional security.

As mentioned above, the average transaction rate is about 8.88 dollars. Although this value refers to the entire dataset, we can see it also makes sense when looking only at the positive values - since there are more 1$ frauds than all the other sums combined.

Part 3. - Clustering

As explained at the beginning, we use several clustering algorithms and measure our success. We used the values of Recall, Precision, and F-1 Score.

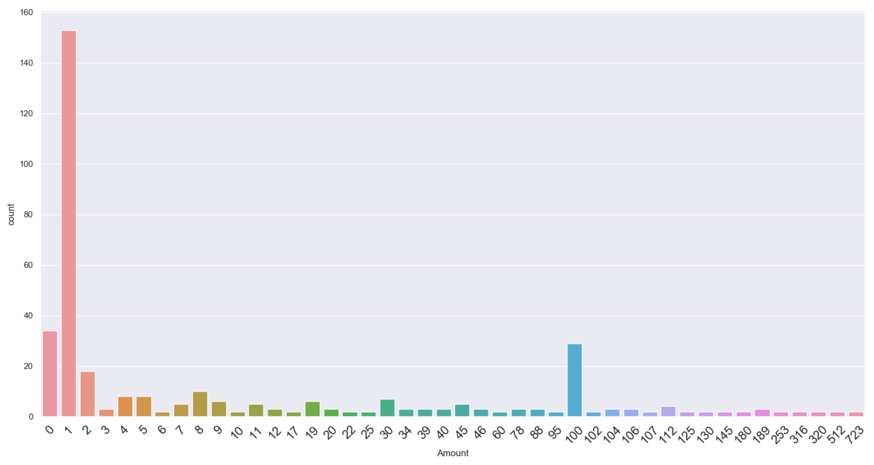

FIGURE 2. Histogram of the frauds over different amounts (with 2 transactions or more)

1. K-MEANS

Our first shot is the most primary clustering algorithm - K-Means. We chose not to count the "Amount" column as input for the algorithm, because it does not result high correlation with the "Class" column (see pt. 2). Obviously, as mentioned above, we seek for 2 clusters - for a binary classification. Moreover, since clustering all the data gives terrible indices (precision 0.000994, recall 0.272000 and f-1 score 0.001981), we decided to sample randomly the data, so that the ratio between positive and negative samples would be more reasonable.

First, we made an even sampling and got the results. But then, we wondered whether they could have been improved, if we had sampled more negative examples. Surprisingly, there has been an increase at the f-1 score until ratio of 100, i.e. 492 positive example with 49,200 negative ones. With ratio=1 we got f-1 score 0.4907, and with ratio=100 we got 0.539 - a 5% improvement. The various attempts are detailed at **table 1**, and the best result's clustering (ratio=100) is shown on **figure 3**.

Since fully representing 28-D data is impossible, and the variances of V1 and V2 are the highest of all the PCA features, we project it on those axes. In order to get the meaning of the results, we attach on the left side the real labels of the data, projected on dimensions V1, V2 as well.

The f1 score of kmeans in randomization is between 0.0 and 0.01749.

2. FUZZY C-MEANS

Fuzzy C-Means (abbreviated as FCM) showed better results, especially on the Recall value. This is due to the fact that the clustering is more "soft": the labeling of each point is not strictly 0 or 1, but 0 in some probability, and 1 in its complement ($\mathbb{P}(label = 0) + \mathbb{P}(label = 1) = 1$). Hence, the algorithm can classify more relevant items as positive, more softly - which is the definition of increasing the Recall value. As we learned at class, the FCM algorithm has a hyperparameter m, which determines the level of fuzziness. A value closer to 1 is more "hard", while higher value is more "fuzzy". We wrote a function called "cutoff" to classify the points whose probabilities are higher than some threshold. We tested the algorithm's performances over different m values, and compare the 3 indices (Recall, Precision, F-1 Score). Similarly to K-Means, we adjust the ratio of sampling, because we encounter

	precision	recall	f-1 score
all the data	0.000994	0.272000	0.001981
evenly sampled (ratio=1)	1.000000	0.325203	0.490797
ratio=2	1.000000	0.333333	0.500000
ratio=5	1.000000	0.343495	0.511345
ratio=20	1.000000	0.367886	0.537890
ratio=50	0.989130	0.369918	0.538461
ratio=100	**0.978609**	**0.371951**	**0.539028**
ratio=125	0.943005	0.369918	0.531386
ratio=140	0.942708	0.367886	0.529240
ratio=200	0.002508	0.237804	0.004964

TABLE 1. Comparing different positive-negative sampling ratios in K-Means

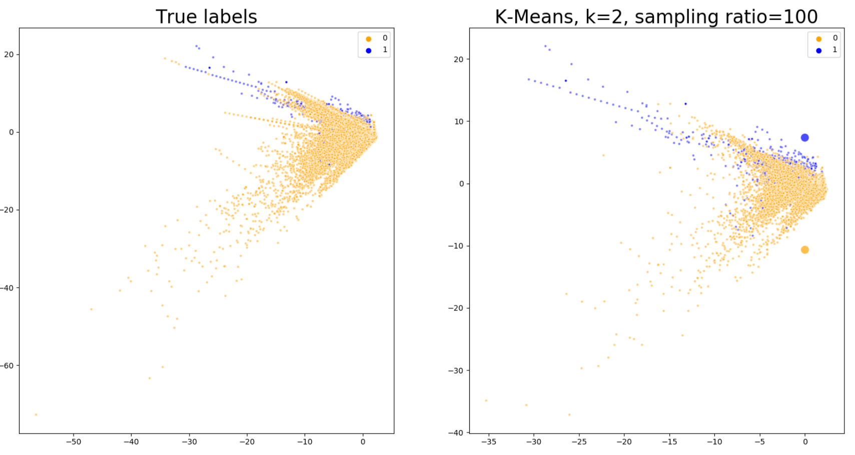

FIGURE 3. Performing K-Means on the 28-D sampled data(without "Amounts") with ratio=100, and projecting it on V1, V2

the same problem as before - using the entire vectors gives the values: precision: 0.002866, recall: 0.806910, f1: 0.005711.

In order to get the biggest picture of the situation, we ran the algorithm repeatedly, with several values of "ratio" and "m": $ratio = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, $m = [1.1, 1.2, 1.5, 1.8, 2.0, 2.5]$, i.e. 60 iterations of the algorithm. The top 10 results are shown on **table 2**, ranked by their F-1 score, and the best of them ($m = 2.5$, $ratio = 7$) is plotted at **figure 4**.

The f1 score of FCM in randomization is between 0.04179 and 0.10447.

	ratio	m	precision	recall	f-1 score
1	**7**	**2.5**	**0.973529**	**0.672764**	**0.795673**
2	6	2.5	0.971875	0.632114	0.76601
3	5	2.5	0.965079	0.617886	0.753408
4	4	2.5	0.986441	0.591463	0.739517
5	2	2.5	0.996212	0.534553	0.695767
6	8	2	0.996154	0.526423	0.68883
7	6	2	0.99177	0.489837	0.655782
8	7	1.8	0.982609	0.45935	0.626039
9	6	1.8	0.991111	0.453252	0.622036
10	5	1.8	0.995413	0.441057	0.611268

TABLE 2. The values of various m and $ratio$ parameters, ranked decreasingly

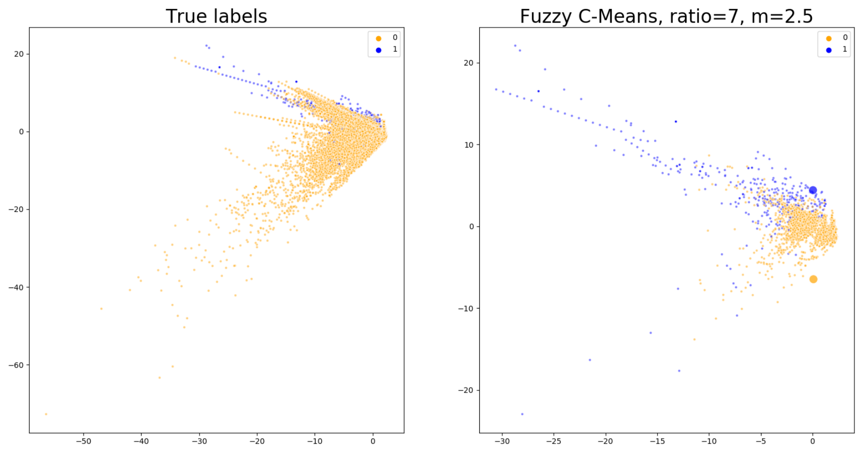

FIGURE 4. The performance of Fuzzy C-Means over a sampled data with $ratio = 7$ and $m = 2.5$

3. GMM

The third algorithm we are using is the "Gaussian Mixture Modeling"(GMM). Unfortunately, it did not get any better results than FCM. The best f-1 score we achieved is 0.568, using sampling rate $ratio = 2$. the top 7 results are detailed on **table 3**.

The f1 score of GMM in randomization is between 0.31391 and 0.40789.

4. Spectral Clustering

The Spectral Clustering algorithm is well-known for its analytic and nice results regarding binary clustering data. It did not disappoint us in our situation either, as it showed the best results of all the algorithms - 88% f-1 score, when performed with the right sampling rate.

ratio	precision	recall	f-1 score
1	0.994	0.35	0.517
2	**0.511**	**0.64**	**0.568**
3	0.15	0.238	0.184
4	0.334	0.642	0.439
5	0.091	0.232	0.131
6	0.083	0.252	0.125
7	0.074	0.26	0.116

TABLE 3. GMM results with several sampling rates

We used the "sklearn.SpectralClustering" class. the input of its "fit" method is a distances matrix (also called "affinity matrix" at the documentation), which we calculated with "scipy.spatial.distance_matrix". Since we did not manage to control the randomness of the results, we ran the algorithm over several ratio values, a few times. Then, we saved the variables respectively to the best f-1 score. The full process is detaild on the method "spectral_main".

Hereby are the results of one activation of the function (**table 4**).

The f1 score of Spectral Clustering in randomization is between 0.19026 and 0.26549.

0	1	2	3	4	5	6	7	8	9	10	11
ratio	1	2	**3**	4	1	2	**3**	4	1	2	3
precision	1	1	**0.968**	0.974	1	0.088	**0.968**	0.991	1	0.991	0.06
recall	0.23	0.23	**0.809**	0.23	0.23	0.191	**0.809**	0.23	0.23	0.23	0.191
f-1 score	0.374	0.374	**0.882**	0.372	0.374	0.121	**0.882**	0.373	0.374	0.373	0.092

Table 4

12	13	14	15	16
4	1	2	3	4
0.971	1	0.088	1	0.991
0.805	0.23	0.191	0.23	0.23
0.88	0.374	0.12	0.374	0.373

Table 4

On **figure 5** one can see the best f-1 score of the algorithm (88%). It was achieved with ratio value of 3.

TABLE 4.

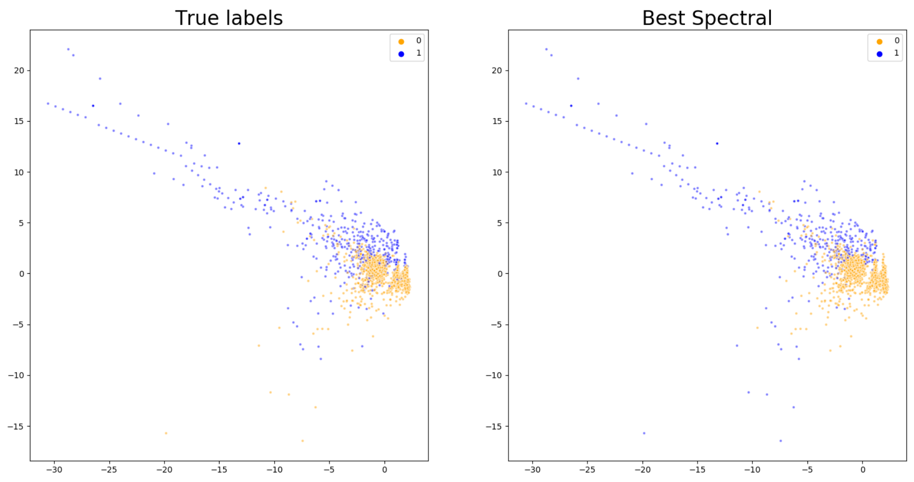

FIGURE 5.

5. DBSCAN

We applied the DBSCAN algorithm, which has two hyperparameters: the radius ϵ in which we look for neighbors, and the number of neighbors k which constitutes a core point. We stipulated that the "noise" points that don't belong to any cluster are classified as frauds (positive), while all clusters are classified as negative. Notice that by increasing ϵ we get more core points and thus more points in clusters, thus less points labeled positive. However, by increasing k, we get less core points, thus more points labeled positive. We experimented with different values of these parameters, while also taking different sample sizes. These are the values attaining the highest f-1 score(**table 5**):

Our best results were achieved with an even sampling, and with ϵ around 4. The following diagram describes the clustering compared to the ground truth:

When taking the whole dataset, even while optimizing the hyperparameters, we got bad results: good precision of 0.9695, but bad recall of 0.00364, making for an f-1 score of 0.007261.

The f1 score of DBSCAN in randomization is between 0.44211 and 0.51789.

Part 4. - Conclusions

From our results we derive several conclusions:

Firstly, the extreme imbalance in the data proved to be a challenge for all the algorithms. The small set of positive samples was hard to identify, and most algorithms labeled a much larger set positive. These results led to the idea of sampling just a small fraction of the negative samples, which indeed improved the performance of all algorithms. Note, however, that an even sampling of positive and negative points (1:1 ratio) wasn't always optimal.

When experimenting with different feature vectors, we found that for some algorithms, the vectors including the amount performed slightly better on the whole data, but didn't make much of a difference when taking a smaller

No.	Ratio	k	ϵ	Precision	Recall	f-1 score
1	**1**	**11**	**4.4**	**0.884**	**0.823**	**0.853**
2	1	17	4.4	0.872	0.831	0.851
3	1	12	4.4	0.879	0.825	0.851
4	1	11	4	0.822	0.882	0.851
5	1	19	4.4	0.827	0.866	0.846
6	1	12	3.6	0.764	0.943	0.844
7	1	19	4	0.767	0.931	0.841
8	1	13	4.4	0.873	0.809	0.84
9	1	16	4.4	0.844	0.833	0.838
10	1	8	4	0.835	0.841	0.838

TABLE 5. Top 10 f-1 scores of DBSCAN parameters

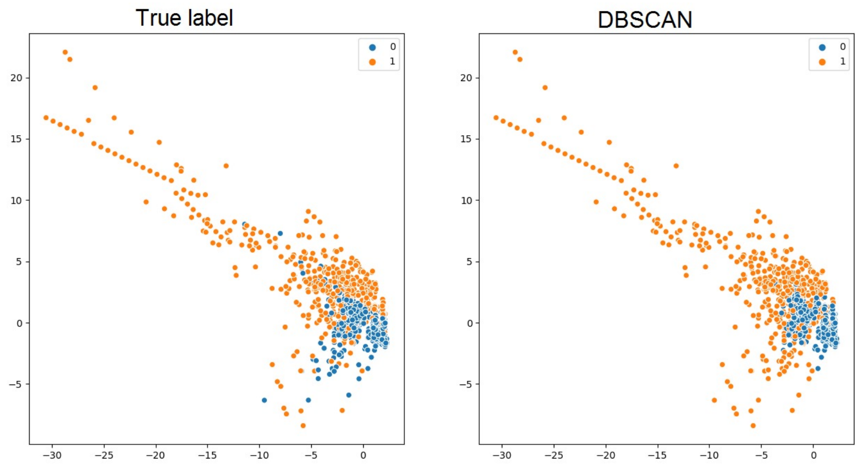

FIGURE 6. The performance of DBSCAN over a sampled data with $ratio = 1$, $k = 11$ and $\epsilon = 4.4$.

sample. From this we can conclude that the amount does provide some information, but gives no strong indication as to whether the transaction is a fraud or not. These also presented a problem of scale: since the amount feature is on a bigger scale than the PCA features, and not distributed normally, this affected the clustering more than the other features. When normalizing the amount, however, it lost its impact.

The time feature was very hard to handle, and we ended up not using it. Its main impact was that at certain times less transactions were made (presumably at night time). But this gives us no indication, as less frauds were made in these times as well. We tried to include the time feature, but like the amount feature, it was on a bigger scale and messed up the clustering.

Overall, DBSCAN and spectral clustering performed better than the Gaussian-based algorithms. We think this is because the data is not linearly separable. GMM and its derivatives are good for data consisting of two distinct normal distributions. In our case (as can be seen by our statistic analysis of the fraud data), the frauds are not distributed normally, but rather behave like noise. Consequently, DBSCAN works very well, because it fits best to the case of dense "regular" data and sparse noise around it. Spectral clustering fits this case as well, because its motivation is to maximize the distances between the clusters. Since the negative data is dense and the noise is far around it, spectral clustering algorithm places them in two different clusters. This explains the success of those two algorithms over the others.

Another problem we faced was the lack of strong correlations between features and class. None of the features gave a strong indication as to the class, which made GMM-based clustering algorithms less effective. This supported our conclusion that the data don't consist of two distinct Gaussians.

Part 5. - References

- the data source, with explanation - `https://data.world/raghu543/credit-card-fraud-data`
- sklearn clustering package - `https://scikit-learn.org/stable/modules/clustering.html`
- sklearn. metrics - measuring the algorithm's success - `https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics`
- seaborn different plotting methods - `https://seaborn.pydata.org/api.html#api-ref`
- fuzzy-c-means: An implementation of Fuzzy C-means clustering algorithm - by Madson Luiz Dantas Dias, 2019, University of Ceara, Department of Computer Science; url: `https://github.com/omadson/fuzzy-c-means`