Assignment 2

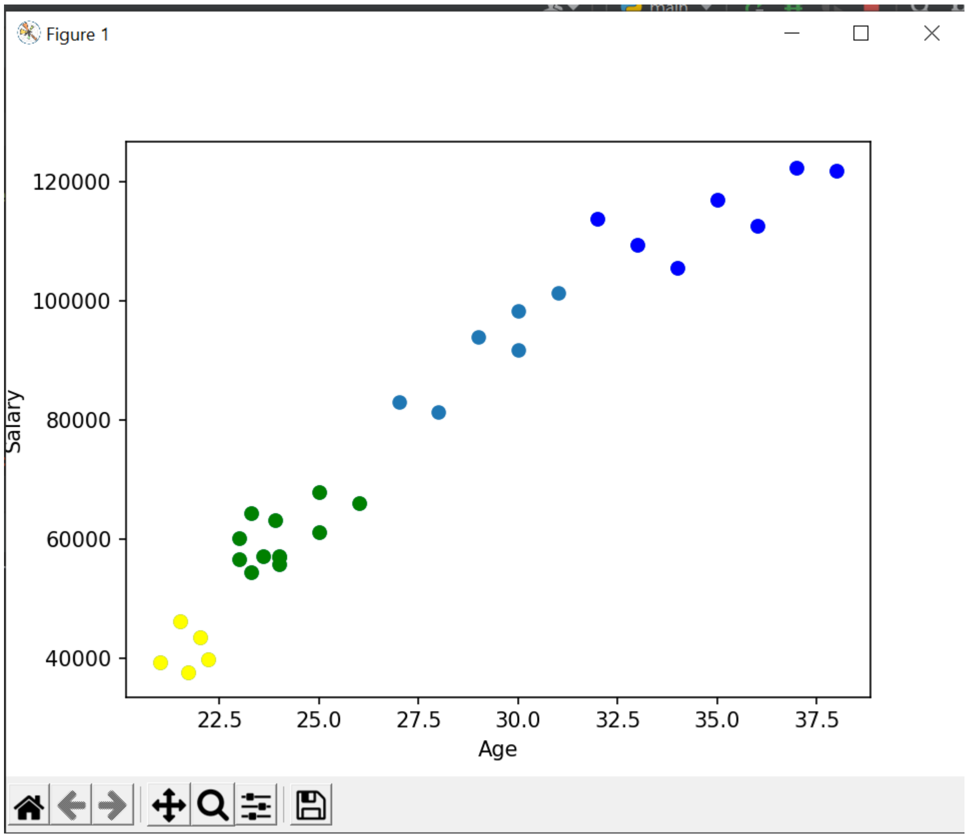
Data mining CSE 454 – Gebze technical university

171044095 – Djuro Radusinovic

Note: In this assignment I used 2 datasets found on Keggle. One is salary according to age and number of years of experience. The other contains data about car prices in the past from various companies. It has much more data than that and contains 28 columns. Data for it is preprocessed and normalized after which I used the clustering algorithm to find clusters in it. I didn't put graphs for DS2 since it wasn't required and due to high dimensionality would be somewhat cumbersome though I did calculate the execution time for it. Also, for FPgrowth I had to use a different dataset because FPgrowth is used for finding associations in transactional datasets meaning its usecase is completely different from other three algorithms.

KMEANS

For this algorithm I used Kmeans algorithm in order to visualize 2 columns from a 3d dataset. For the value of N {number of clusters} 4 was used in this case and as we can see we got the following results in this graph:



*Computational time of this algorithm on **DS1** is about 0.01s*

*Silhouette score for K-means for **DS1** is 0.6563762976553834*

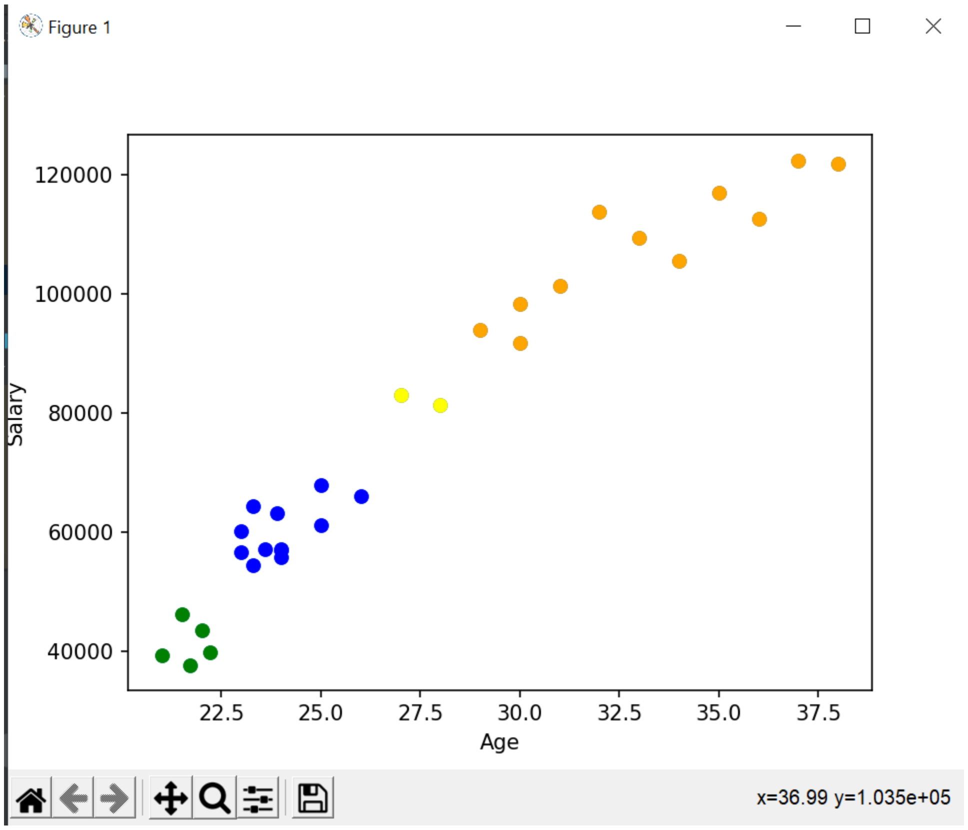
*Silhouette score for K-means for **DS2** is 0. 464388244748624*

*Computational time of this algorithm on **DS2** is about 0.143s*

*Time complexity of K-means for running a fixed number t of iterations of the algorithm takes only O(t*k*n*d), for n (d-dimensional) points, where k the number of clusters. Now we used t as 1 here and used k as 4 so in our case we can regard them as constant meaning it would yield somewhat linear behavior for our case. But K-mean's complexity gets much more comlicated when we try to find the ideal number of K clusters. Since I just took a local optimum there will be no problem and this complexity will hold though as the number of dimensions increase finding K gets more and more difficult.*

DBSCAN

Similar thing was now performed with DBscan algorithm. Again similar library was used in order to achieve this result. It is necessary to understand here that DBscan isn't the best for the datasetes I used here since DBscan is used to recognize various different shapes though it can work in fine in some cases. I would personally prefer to use Kmeans for such numeric data that was used here like Age and Salary. Now its advantage is that it is less sensitive to noise than Kmeans so we can use that even though it is not as efficient in finding appropriate clusters and k-means ended up being much more reliable in this case.



*Average exec. For **DS1** Time was about 0.003s.* | *min_samples = 2, epsilon = 5000* |

*Silhouette score for DB-scan is for **DS1** is 0.6011745984114392*

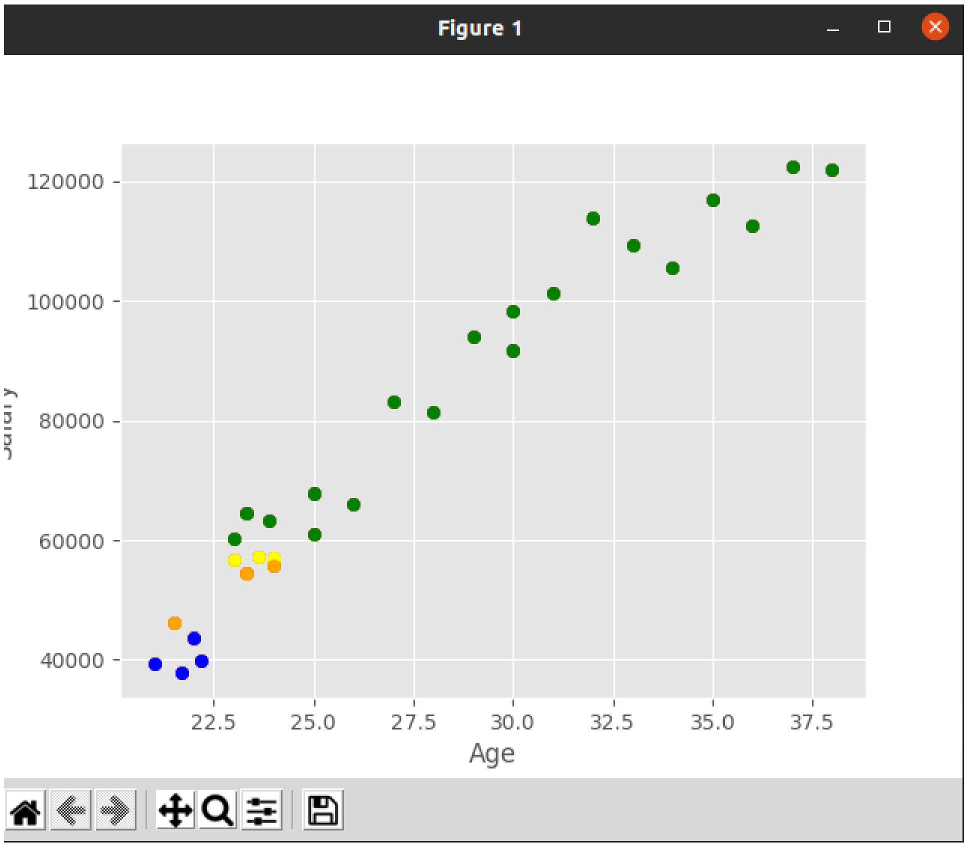
*Average exec. For **DS1** Time was about 0.03240466117858887s* | *min_samples = 2, epsilon = 0.23* |

*Silhouette score for DB-scan for **DS2** is 0.2999828164364935*

Time complexity of this algorithm is O(nlogn)

CHAMELEON

Here I used clustviz's library in order to perform this. As expected for our dataset this algorithm won't work well since there are no shapes here really that would be recognizable.



Average execution time: 0.25301313400268555 *k = 2, knn = 4, m = 6, alpha = 2*

Average execution time: 0.4458177089691162 *k = 2, knn = 4, m = 6, alpha = 2*

*Silhouette score for Chameleon algorithm with **DS1** is 0. 21252762896715563*

*Silhouette score for Chameleon algorithm with **DS2** is 0.35175991491488046*

Time complexity of this algorithm is quadratic, or O(n^2)

Silhouette score overview

After running algorithm for the clusters we can see that the clear winner, or most quality cluastering algorithm for our case is K-means with the score of **0.6563762976553834.**

Following it comes DBscan with the score of **0.6011745984114392.**

Last place is taken by Chameleon with the score of **0.25301313400268555**

For the datasets where more dimensions were included we see that K-means is stll the best performer with the score of **0.464388244748624**, than Chameleon with the score of **0.35175991491488046** and the last place is taken by DBscan with the score of **0.2999828164364935**

These results make perfect sense since this dataset doesn't form shapes which would be suitable for dbscan or chameleon, but it is good in order to perfrom quantization on it and see the clusters with the nearest mean values.

COMMENT ON ALGROTIHMS

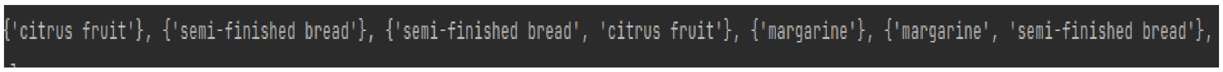
Here we can see that the ideal performer for our dataset was K-means algorithm. Both DBscan and Chameleon have worse SiIlhouette coefficients and we have previously seen that their time complexity is not too good. The other thing is that it is not as easy to predict parameters we would use for those algorithms. In K-means finding K is as easy as finding the Elbow using Elbow method. Since datasets I used do not form shapes which is the real life use for DBscan and Chameleon (recognizing shapes), they weren't able to perform as well as K-means algorithm. Also time complexity is better for K-means in our case than it would be for either DBscan or Chameleon. Of course, both DBscan and Chameleon are very powerful and their execution time is not too far from other algorithms. But, these datasets can get much larger and at that point we would see the difference that would happen.

FPGROWTH

For finding association using FPgrowth algorithm I used transactions.csv dataset. We have nominal attributes in forms of grocery products in this datasets making it ideal for use here as it is a transactional dataset.

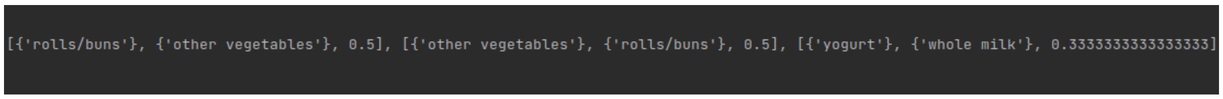
From this dataset I was able to extract its frequent itemsets and association rules.

Since it is too long I am attaching just some of the above mentioned.



```
{'citrus fruit'}, {'semi-finished bread'}, {'semi-finished bread', 'citrus fruit'}, {'margarine'}, {'margarine', 'semi-finished bread'},
```

Some frequent itemsets



```
[{'rolls/buns'}, {'other vegetables'}, 0.5], [{'other vegetables'}, {'rolls/buns'}, 0.5], [{'yogurt'}, {'whole milk'}, 0.3333333333333333]
```

Some association rules

Execution time of this algorithm for the given dataset was: 0.0028874874114990234

*Time complexity of FPgrowth is O(n^2) { items in header table * maximum depth of tree }. This algorithm has big advantage over the apriori of course as it doesn't iterate over all the candidates.*