Machine Learning ex3

Michael Mitsios (cs2200011)

May 2021

2.1 Implementing K-Means in Python

2 classes where needed in order to implement the k-means with restarts. The first one is **k_means_model** and the **k_means.**
k_means_model:

- **Init function** is the constructor of the class. There should perfom the initialization method that is asked. This is why we also give as parameter the data. In a few words the Init function will be called when the class is defined and will initialize the k and will compute the first set of centroids that k-means will use. Based on the value of init_method the Init will either define some random centroids or follow the k-means++ methodology.

- **Fit**: The logic behind this function is to rearrange the centroids (find the real centroids of the clusters) and if the classification of all the points does not change then we have found the best possible solution.

k_means: This class is used in order to use the num_restarts. This class creates many models of the previous class and keeps the one with the smallest objective value. Then it keeps the best performing centroids and classification.

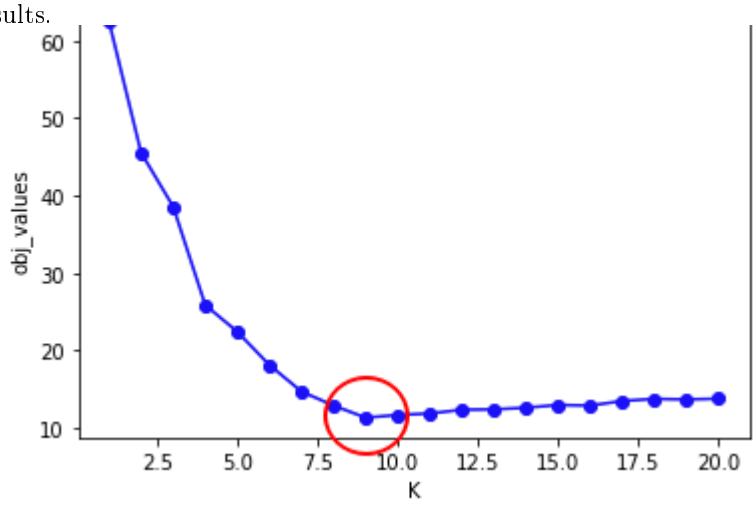
- **fit_best_model:** This function is where we loop over the num_restarts models and initialize and fit each one of them individually. We keep a min value of the objective value each time and when we find one value that is lees than the value we already have we replace it among with all other sets.

- **predict:** This function predicts and finds the centroids that a set of points belongs to. It uses the self.centroids that fit_best_model produced.

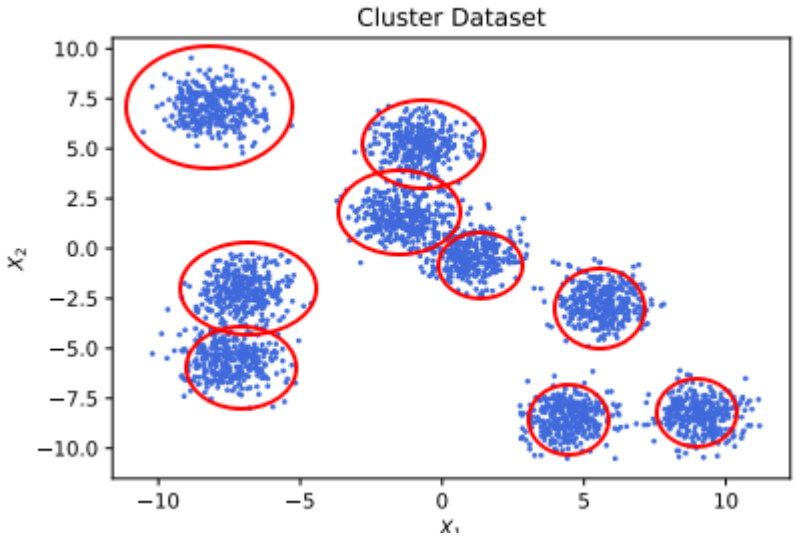
In both classes we have a function called obj_function. This function is the value we want to minimise. The less the value, the best performance our model have, and the better centroids have been found.

2.2.1 The effect of K

In this section I run a for loop for each of the kappas [1,20]. For all the models the num_restarts was set to 10 and the init_method was the k-means++ approach. After creating the model, I evaluated it based on the objective value and I kept the scores in a list. After running all the different values of K, we plot the results.



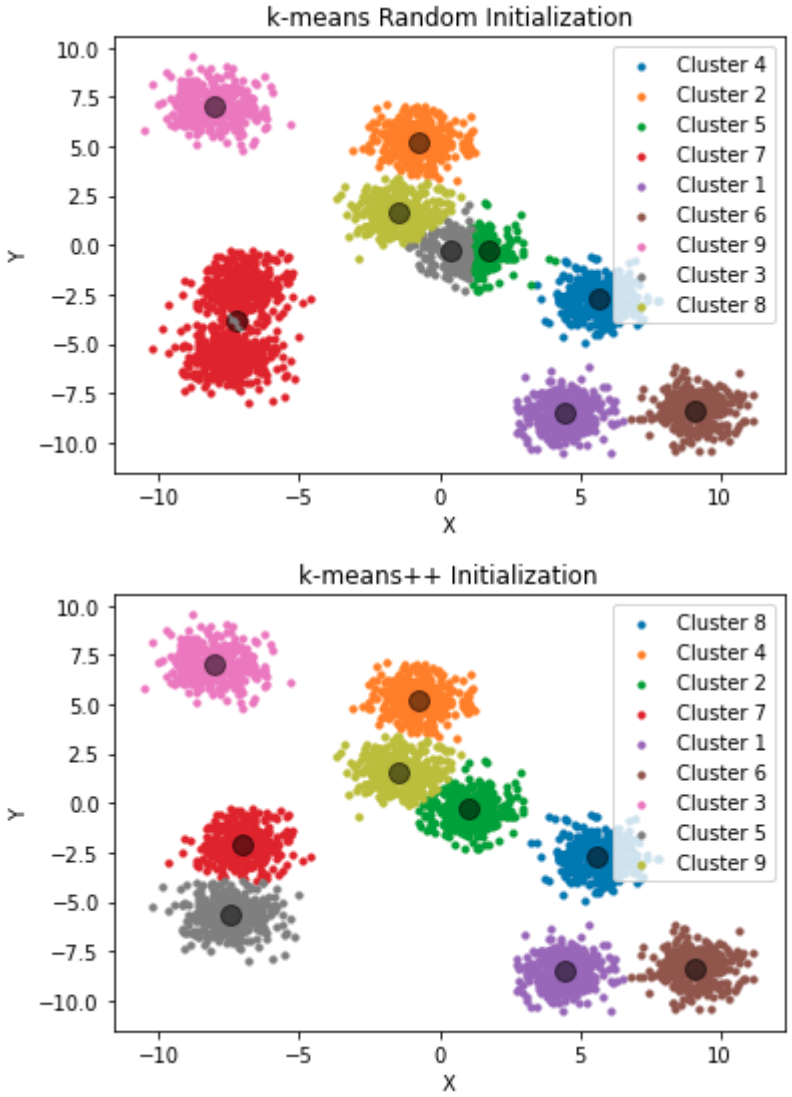
A heuristic approach to selecting a value for k is to investigate the results and select the value at the "bend" of the plot. Increasing the K more than this point the decrease in the objective value will be very little. This is where the K takes our optimal value. As we can see from our diagram the best value of K is somewhere between [8,10] and more specifically seems to be k=9. This value of K agrees with our dataset because as we can see:



We can count around 9 individual clusters. So this technique and our conclusion is a valid one.

2.2.2 The effect of initialization

In a first look we can see that the k-means++ approach clusters better our data than the random approach. 9 Clusters and 10 restarts for each model.



Note: The Clusters in legend are unordered because they come from a dictionary.

k-means clustering is obviously better than the random initialization. This is happening because the k-means++ actually prefers the next center to be as

far as possible from the others 2. The first centroid it picks is random and then afterwards it selects one of the points as the next centroid that has a very big distance from the cluster it belongs. In a few words it takes one of the furthest points from the already defined centroids.

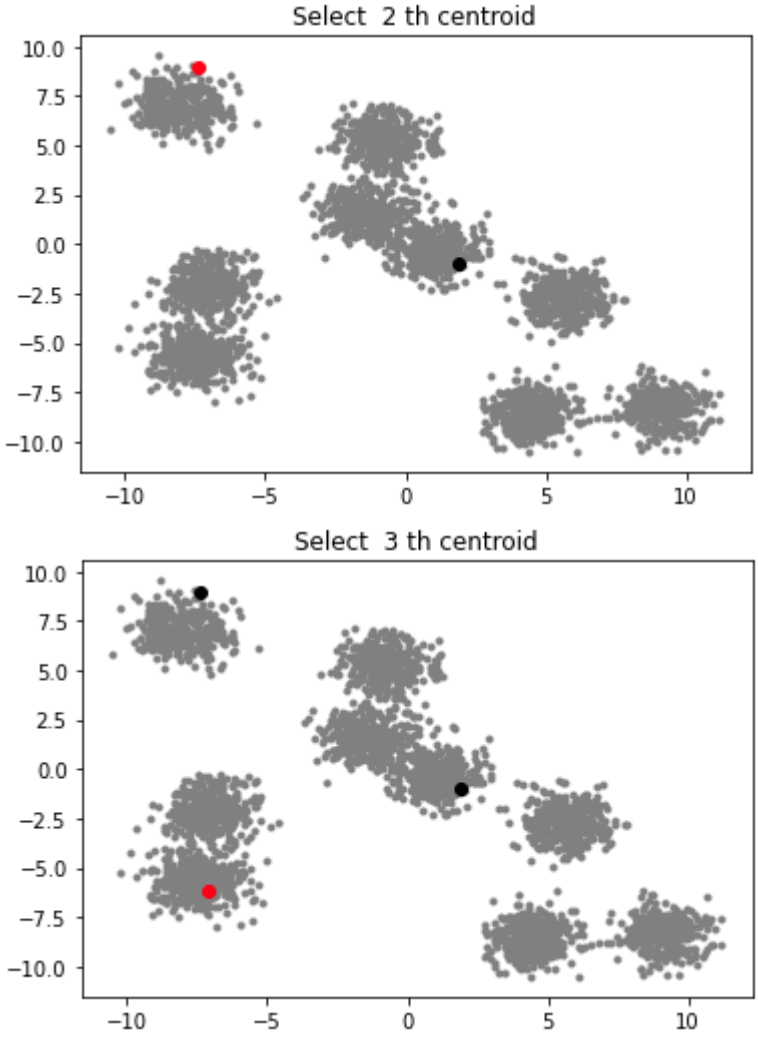
This can be more clearly when we run the 800 runs with 1 restart. In this experiment we can clearly see that the k-mean++ performs way better and more consistent than the random initialization.

	k-mean++	random
mean	16.377851265658265	13.17725769015338
standard deviation	1.8525222588844168	1.7893893169087467

First of all the mean statistic of the objective values is way smaller that the random. This means that the data points are clustered better (closer) to the selected centroids. On the other hand, the standard deviation of k-means++ is also smaller than the one on the random method. In other words, the points seem to be a bit closer to the centers more consistently. For example, the set of [0,0,14,14] and [6,6,8,8] has the same mean **7** but the left one has values closer to the mean, and it has also less standard deviation.

Problem with random initialization

The random initialization would choose at random among the data points the centroids. By following this technique the initial centroids are leading to a non-optimal clustering. The reason why an initial centroid can lead to a not optimal solution is that after the initialization we just apply adjustments to the already selected centroids. The changes we are going to make are to position the centroid to the optimal position in the near are. The rearrange re make cannot take a centroid and place it where a centroid is missing. This is why the initialization is **VERY** important. To solve this problem, we are using the k-means++ method. This method uniformly random picks the first centroid and in order to select the second one checks the distances of the points in data, compared with the 1st centroid. The points that are furthest have the biggest possibility to be the next centroid. After that we just classify the points to the nearest centroid, and find the points with the biggest distance to the centroid they belong would be more possible to be picked as the next centroid.



The red is the new centroid and the black dots are the old ones, among our data. The legend is omitted because it hides information.

In a few words the next centroid that we are going to select, has a very high probability to be farther from all the other centroids.

3 Kernel Construction

Proofs:

1.$k(u,v) = c_1K_1(u,v) + c_2K_2(u,v),\ c_1, c_2 \geq 0$

First the $c_1K_1(u,v)$ is a valid kernel function because:

$$c_1K_1(u,v) = c_1 < \varphi_1(u), \varphi_1(v) >=$$

$$c_1K_1(u,v) =< \sqrt{c_1}\varphi_1(u), \sqrt{c_1}\varphi_1(v) > (1)$$

which has the form of an inner product (so it can be expressed like one).

From (1) we have:

$$k(u,v) = c_1 K_1(u,v) + c_2 K_2(u,v) =$$

$$k(u,v) =< \sqrt{c_1}\varphi_1(u), \sqrt{c_1}\varphi_1(v) > + < \sqrt{c_2}\varphi_2(u), \sqrt{c_2}\varphi_2(v) >$$

$$k(u,v) =< \left[\sqrt{c_1}\varphi_1(u)\sqrt{c_2}\varphi_2(u)\right], \left[\sqrt{c_1}\varphi_1(v)\sqrt{c_2}\varphi_2(v)\right] >$$

and we see that k(u, v) can be expressed as an inner product, so its a valid kernel function (via Mercer's theorem).

2. $k(u,v) = K_1(u,v)K_2(u,v)$

By using the Mercer's theorem, since $k_1$and $k_2$are valid kernels they must be represented by an **inner product**. So let a be the feature vector of $K_1$and b the same for . So we have:

$K_1(u,v) = a(x)^T a(y)$, where $a(z) = [a_1(z), a_2(z), ...a_M(z)]$ we assume that a produces an M-dim vector.

AND$K_2(u,v) = b(x)^T b(y)$, where $b(z) = [b_1(z), b_2(z), ...b_N(z)]$ we assume that b produces an N-dim vector.

$$k(u,v) = K_1(u,v)K_2(u,v)$$

$$k(u,v) = \left(\sum_{m=1}^{M} a_m(x)a_m(y)\right)\left(\sum_{n=1}^{N} b_n(x)b_n(y)\right)$$

$$k(u,v) = \sum_{m=1}^{M}\sum_{n=1}^{N}\left[a_m(x)b_n(x)\right]\left[a_m(y)b_n(y)\right]$$

we define c(z) as a M·N-dimensional vector, where $c_{mn}(z) = a_m(z)b_n(z)$

$$k(u,v) = \sum_{m=1}^{M}\sum_{n=1}^{N} c_{mn}(x)c_{mn}(y)$$

$$k(u,v) = c(x)^T c(y)$$

So k is also represented as an inner product so the $k(u,v)$ is a valid kernel function(via Mercer's theorem).

3.$k(u,v) = f(k_1(u,v))$, where f is a polynomial function with positive coefficients.

Since each polynomial term is a product of kernels with a positive coefficient, the proof follows by applying 1 and 2 proofs.

4.$k(u,v) = exp(k_1(u,v))$ This form is exactly the same as 3(polynomial), because:

$$exp(x) = \lim_{i->\infty}\left(1 + x + ... + \frac{x_i}{i!}\right)$$

In combination with:

$$k(u,v) = \lim_{i->\infty} k_i(u,v)$$

The k(u,v) is a valid kernel function.

4 Support Vector Machines

First I normalized the data into [0,1] normalization and [-1,1] normalization. For the first one I used the **MaxAbsScaler** of sklearn. I fit_transform my train data and then just transform the test data. For [-1,1] normalization I used the **MinMaxScaler** where I can define the range I want. In both cases I print the first result in order to see that all went well. In my opinion one of the main reasons we normalize our data is to make easier the performance and the calculation. Because we are going to do many fits in order to find the best parameter combination we need to make as simple as possible the calculations and the metrics finding.

For the second question I used the function GridSearchCV we used in the lab. In order to run faster I made 2 critical changes:

1. **Set the cv to 5:** In order to return the best parameter combination, the GridSearchCV has to validate all the combinations and sort them. To validate each one of them and have some metrics it uses cross-validation10. The cross-validation in its own is a very good technique but if we make 10 folds the whole process becomes very complex. This is why I set the cv parameter to 5 in order to make the half crosses. In this way we still have a very good method to judge each parameter combination, but also the whole process will be done a lot quicker.

2. **Set the parameters as a list:** At first I tested the process with a small sample to check its behaviour. If we have a set of parameters like this:

```
params_={"C": [0.1, 1, 10, 100],
         "degree":[1,5,10],
          "gamma": [0.01, 0.001,'scale','auto'],
          "kernel":['linear', 'poly', 'rbf', 'sigmoid']}
```

We are going to have around 430 different combinations of parameters. But if we take a closer look there are combinations that we don't care about. For example the parameter **degree** is only used from the **poly** kernel! This means that for each of the other kernels will be ignored. There are garbage combinations if we use this form of parameter_grid. We don't care to calculate the {'C': 0.01, 'degree': 1, 'gamma': 1, 'kernel': 'linear'} and {'C': 0.01, 'degree': 5, 'gamma': 1, 'kernel': 'linear'}, because the linear kernel does not care about the degree parameter and the same results will be returned.

To fix this I organized my parameters in a way to avoid useless calculations among them.

```
params_=[{"C": [0.1, 1, 10, 100],"kernel":['linear']},
         {"C": [0.1, 1, 10, 100],"gamma": [0.01, 0.001,'scale','auto'],
          "kernel":['rbf', 'sigmoid']},
         {"C": [0.1, 1, 10, 100],
         "degree":[1,5,10],
          "gamma": [0.01, 0.001,'scale','auto'],
          "kernel":['poly']}
        ]
```

This will lead to 85 combinations in total. This means that the GridSearchCV will have to calculate 4 times less parameters combinations. In this way the linear kernel would ignore the gamma parameter because it does not use it, and the degree parameter will only take place **ONLY** when the kernel is poly.

Having said that I take these results:

	best parameter combination	**time of fit**
MaxAbsScaler[0,1]	{'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}	27734.671 s
MinMaxScaler [-1,1]	{'C': 100, 'degree': 10, 'gamma': 0.001, 'kernel': 'poly'}	15701.155 s

	train_set that used	**accuracy for train set**	**accuracy for test set**
MaxAbsScaler[0,1]	15000	98%	97%
MinMaxScaler [-1,1]	15000	98%	98%

Observations:

- One observation that I have is that the time of [0,1] normalization was almost twice the time that [-1,1] took.
- The whole results report are into the 2 csv where I marked with green the lines with the best performance.
- I used the model that fitted with the 15k of data and checked for its accuracy. Then I will also train 2 other models using the full data and combine them with the optimal parameters.

	train_set that used	**accuracy for train set**	**accuracy for test set**
MaxAbsScaler[0,1]	60000	99.995%	98.33%
MinMaxScaler [-1,1]	60000	100%	98.64%

fit_time
206.6961007118225
362.5530984401703

As we expected training the model with the full train set will give us better accuracy for the test set.

PCA

In the PCA I had a problem with the combination {'C': 100, 'degree': 10, 'gamma': 0.001, 'kernel': 'poly'}. Even though I was getting very good results

with the original form of the data, after doing the PCA showed that in a way "broke" the model. The time to complete the training was around **half an hour,** and the accuracy was not more than **12%** (both train and test set). I didn't understand exactly why was that bad. Probably the constant values of C and gamma couldn't adapt to the smaller dimension dataset. As a result I abadonded this parameter combination and searched for a new one in order to continue my experiments. I wanted to keep the kernel type the same ('poly') with the previous combination.

In my results as **ranked 2** I had this combination:**{'C': 1, 'degree': 10, 'gamma': 'scale', 'kernel': 'poly'}**. This is the combination I chose in order to continue my experiments (after having a quick check with the PCA).####**The orange line**####

	train_set that used	**accuracy for train set**	**accuracy for test set**
MinMaxScaler [-1,1]	60000	100%	98,64%

fit_time
302.28412413597107

As we can see there is not so big difference on the final accuracy. So we choose this combination of parameters.

Having said that we have to examine the behaviour after transforming our dataset using PCA.

[0,1]

PCA	Components	Train Acc	Test Acc	Fit time
0.95	154	100%	98.54%	97.0656213760376
0.80	44	100%	98.54%	53.293978691101074
0.60	17	99.96333%	97.25%	26.168682575
0.50	11	98.9483%	94.65%	32.374810

[-1,1]

PCA	Components	Train Acc	Test Acc	Fit time
0.95	154	82.815%	71.33%	884.3914382457
0.80	44	86.7183%	83.03%	374.6982681751251
0.60	17	81.8183%	80.59%	114.759517431
0.50	11	78.46%	77.26%	100.972646

The general idea behind the PCA is the **trade off between time and Accuracy**. In a few words, the less dimenssions we have the less the accuracy we will eventually have, but the faster our fit will be done.

But as we can see this isn't the rule. In [0,1] the first 2 PCAs have better performance than our model without PCA! This may happen because we "focus" more on the information needed. Also for some reason in the [-1,1] the time that the train takes grow a lot bigger, but still as we decrease the dimensions the times goes down as well as the accuracy of the model.