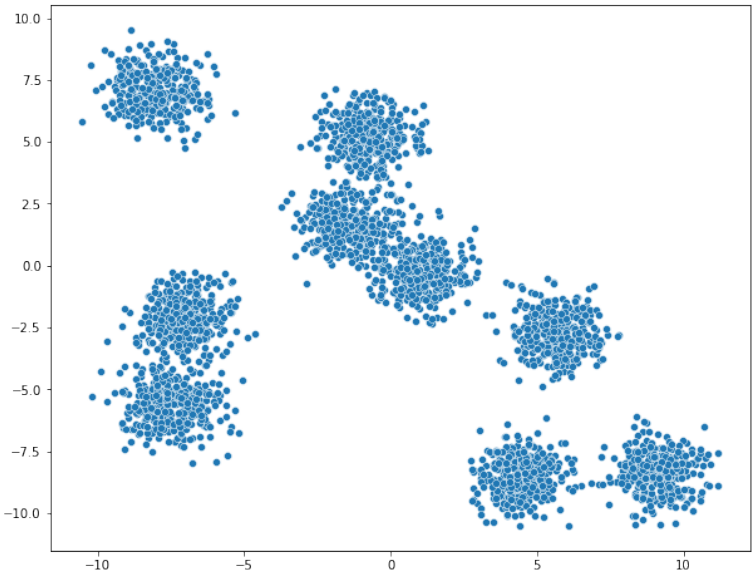
M124 Coursework 3

Vangelis Tsiatouras
cs2190021@di.uoa.gr

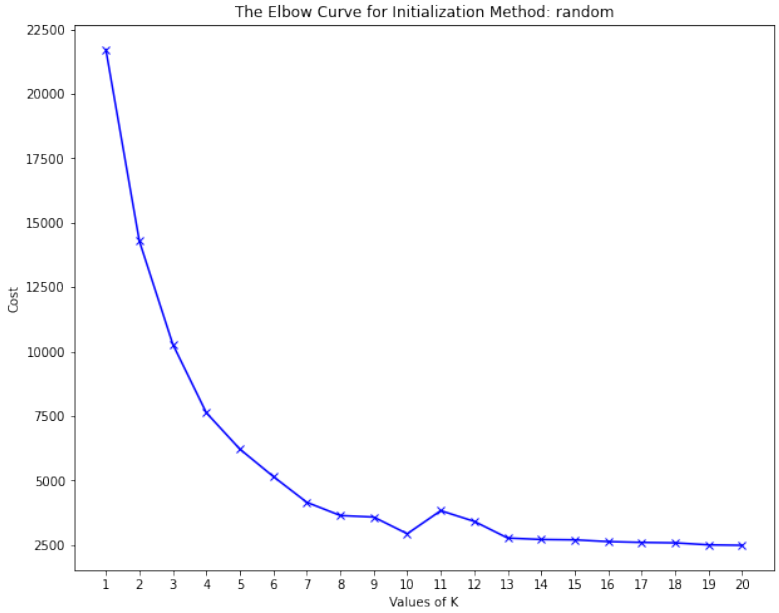
May 20, 2021

K-means Clustering

In this task we have to partition the following observations into clusters. I have implemented the two initialization variations of K-means: Random, K-means++.

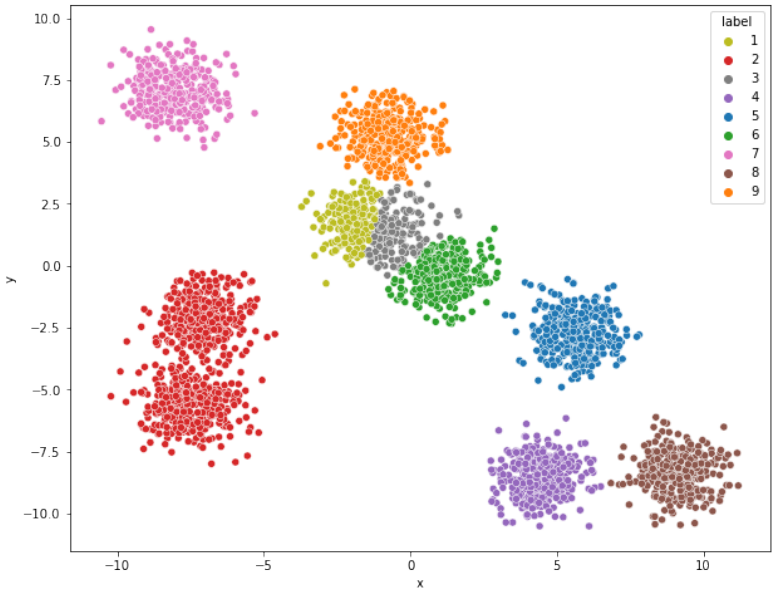


The following graphs visualize the objective function for the two initialization methods and for $k = [1, 20]$. The objective function basically is the sum of the distances of the points from their nearest centroid. This number indicates the 'cost', of how much far are the datapoints from their centroids. So, by having high values of cost it is assumed that we have many datapoints far away from the centroids, contrast to having low values of cost it indicates that the datapoints are near to their centroids and probably split correctly into clusters.

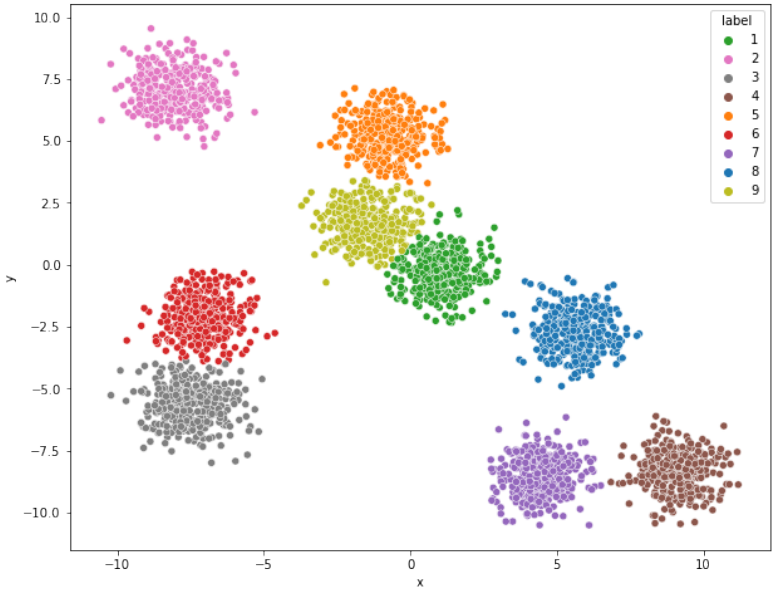


As it can been seen in the graphs above, the bend of both curves it is located for $k = 9$, which intuitively seems correct. Although Random init method yields higher cost values for $k = [10, 13]$ contrast to kmeans++ which has a much more stable partitioning for all ks. Generally, K-means++ theoretically can converge to centroid in less iterations (which in this case is true). But the execution time of Random init is way faster than K-Means++. The big difference of both methods can be observed when we have few iterations (i.e. 10) which K-means++ is much more reliable. For $iterations > 20$ both methods yield the same clusters.

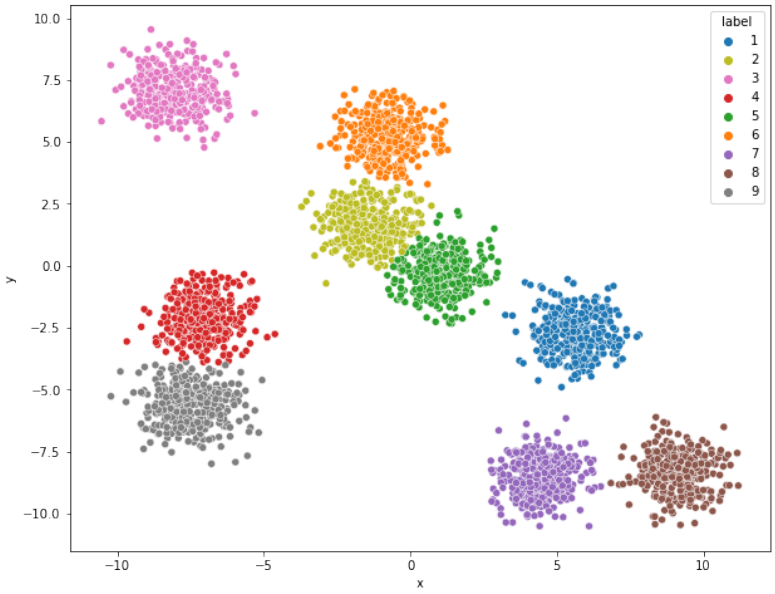
Number of Iterations = 10, Random



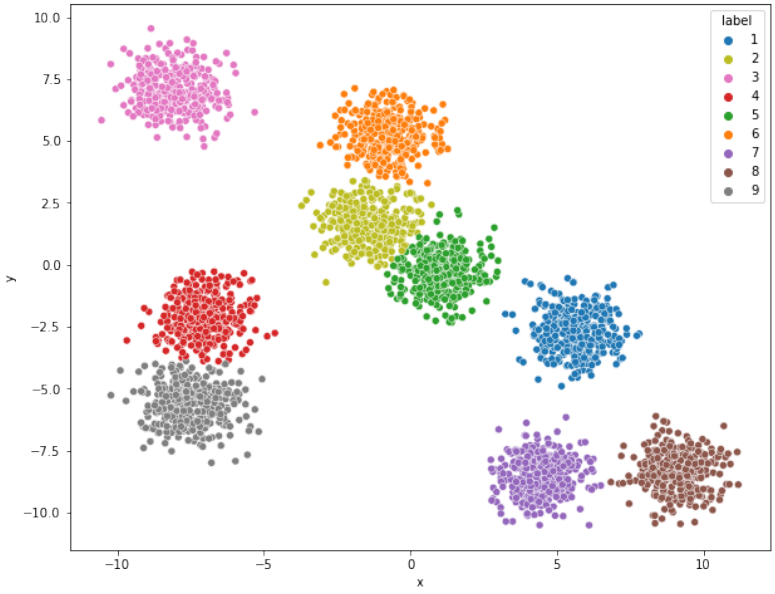
Number of Iterations = 10, K-means++



Number of Iterations = 30, Random



Number of Iterations = 30, K-means++



Support Vector Machines

For this task in order to conclude to best hyper-parameters of SVM model, it was necessary to use Grid Search. After normalizing the dataset to two separate datasets scaled at (0,1) & (-1,1), the best hyper-parameters that yielded are:

(0,1): kernel='rbf', C=10, gamma=0.01 with score: 0.9813333333333334
(-1,1): kernel='rbf', C=5, gamma=0.01 with score: 0.9842666666666666

The scores above refer to accuracy and they extracted after 5-fold cross validation inside the Grid Search.

Running the SVM model with kernel='rbf', C=5, gamma=0.01 for the (-1, 1) normalized dataset, the stats for the test set are the below.

```
Elapsed time: 479.74951577186584 secs
RBF Kernel for (-1,1) normalization
Accuracy: 0.9777
Confusion Matrix
 [[ 988    0    1    0    0    3    3    0    1    0]
 [   0 1134    2    0    1    0    0    2    1    1]
 [   1    0 1020    2    4    0    2    5    5    1]
 [   0    1   11  973    0    9    0    4   13    2]
 [   1    1    1    0  946    1    1    4    0    7]
 [   0    1    3    7    1  838    7    0    5    1]
 [   5    0    0    0    3    5  974    0    2    0]
 [   1    3    7    0    9    1    1 1033    0    9]
 [   0    5    3    5    2    2    3    0  937    6]
 [   1    1    1    4   13    4    1    9    1  934]]
```

The model is pretty accurate to the predictions, but the train execution time is a bit too much. So, let's introduce PCA in order to reduce the execution time, with the cost of losing some small percentage of accuracy.

Initially, I tried PCA transformation with a large amount of number of components (0.95). The results are the following:

```
(60000, 327)
Elapsed time using 327 components: 2230.0537192821503 secs
RBF Kernel for (-1,1) normalization
Accuracy: 0.8934
Confusion Matrix
 [[ 925    0   65    1    0    1    3    0    1    0]
 [   0 1120   17    0    0    0    0    2    1    1]
 [   1    0 1028    1    1    0    0    3    5    1]
 [   0    0  149  842    0    6    0    4   10    2]
 [   1    0   96    0  857    0    0    2    1    5]
 [   0    0  108    9    0  736    5    2    2    1]
 [   2    0  135    0    1    5  844    0    2    0]
 [   1    4  170    0    4    1    0  877    0    7]
 [   0    3  103    7    0    6    1    0  840    3]
 [   0    0   74    7    8    1    0   12    2  865]]
```

With total 327 number of components the performance was hit dramatically, both training time increased and accuracy dropped.

Although, with less number of features, PCA eventually, yields better results. The training time reduced significantly and also for 0.35 number of features the accuracy is acceptable (96%).

```
(60000, 78)
Elapsed time using 78 components: 290.6317615509033 secs
RBF Kernel for (-1,1) normalization
Accuracy: 0.9543
Confusion Matrix
 [[ 967    0   21    1    0    1    5    0    1    0]
 [   0 1125   10    1    0    0    0    2    2    1]
 [   3    0 1027    2    1    0    0    3    2    2]
 [   0    0   41  951    0    5    0    3   11    2]
 [   1    0   41    0  910    1    0    5    0    4]
 [   1    0   21   11    1  820    5    2    2    0]
 [   2    0   50    0    1    5  931    0    0    0]
 [   0    2   58    0    2    0    0  996    0    6]
 [   0    1   40    6    0    2    2    2  905    5]
 [   0    1   28    8    9    1    0    9    2  911]]


(60000, 17)
Elapsed time using 17 components: 29.632370233535767 secs
RBF Kernel for (-1,1) normalization
Accuracy: 0.9665
Confusion Matrix
 [[ 990    0    0    2    0    1    2    0    1    0]
 [   0 1129    3    2    0    0    0    2    2    3]
 [   2    0 1010    8    3    3    1    4    6    3]
 [   1    4   18  946    1    9    1    6   22    5]
 [   1    0    3    1  932    0    0    4    2   19]
 [   2    1    2   11    2  828    7    1    7    2]
 [   3    0    1    0    2    8  974    0    1    0]
 [   1    3    8    4    3    1    0 1029    1   14]
 [   2    7    3   15    1   16    4    3  904    8]
 [   0    0    2   11   13    4    0   11    5  923]]
```

Concluding, PCA with 0.35 number of features decreases the dimensions and therefore the complexity of the problem very efficiently, with the cost of approximately 1% worst accuracy score. In my opinion I would prefer using SVM + PCA(0.35) contrast to simple SVM without PCA transformation, because the training procedure is way faster and also the trade-off of losing 1% of accuracy is minor. Although, for benchmarks the SVM without PCA could be the best choice, in order to squeeze out every last bit, keeping in mind that more sources (CPU/RAM) will be used.