Name: Teja Sajja | Miner2 - itachi
GNumber:G01395837
Part-1 :- Rank: 41| V- Score :- 0.95
Part-2 :- Rank: 32 | V-Score :-0.79

K-Means Clustering

In this assignment we need to implement the K- Mean Clustering algorithm. Assignment is divided into two tasks where in the first part we were given Iris data just to test our algorithm. For part-2 we were given an image dataset which needed to be clustered and evaluated on various k values.

K-means algorithm:

1. We need to choose k centres randomly
2. We repeat step 3 and 4 until we get proper centres assigned to each record until same cluster reappear
3. Then we have to calculate the distance between each record and the above selected centres and find the nearest centre for that record.
4. At the end we update the centres by calculating the mean of all the records or points assigned to it.

Selecting initial centres:

As per the algorithm we need to select the initial centres randomly and then update them for each iteration. Due to the random property in this step it is affecting our algorithm in some cases. I.e, If we choose points which have minimum distance between themselves as initial centres.
It affects our accuracy as it might ignore other clusters and focuses on these near points. To avoid such cases I tried selecting in a different way. Initially I only selected one centre randomly and then selected the rest based on the maximum euclidean distance.

Evaluation methods:

As this algorithm has random property and the k- parameter might vary to obtain a better results we perform different evaluation on the algorithm

- Variance_evaluation:

For this we need to calculate the variance of each data point. This is just to determine if all the points in a cluster are close to each other or not. I used this evaluation method by implementing the k-means in a for loop with max iter and collecting all the outcomes and variances for each iteration and then picking the best result which has the least variance.

- Sum square error evaluation:

In this method I directly use the distance and the square to calculate the error.
I didn't use this evaluation method for every iteration. I only used to see the error rate for various k values.

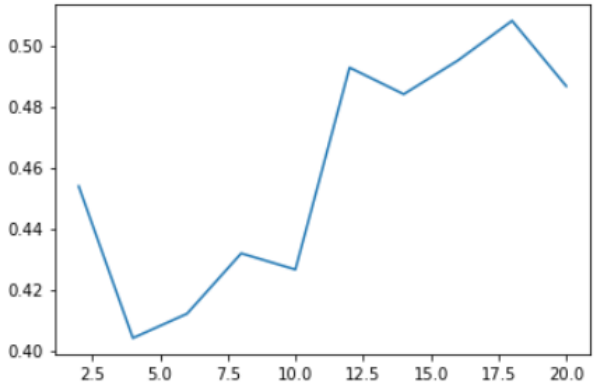
Part-1 : Iris Clustering

- In this part we were given a simple dataset with only 4 features. I implemented the k-means on it. Here there are only three clusters mentioned so My k value was 3. I have initialised the cluster as mentioned above which was very helpful. If I directly

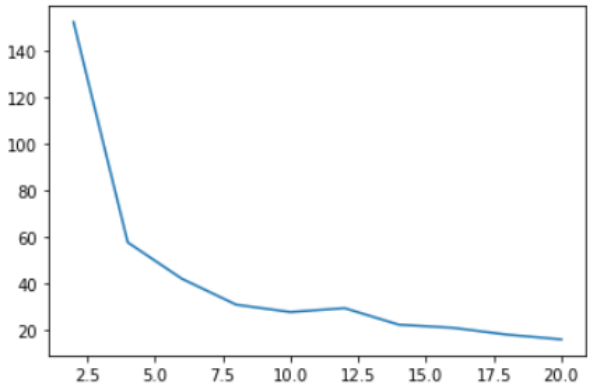
the three clusters without considering the distance among them there was a drastic variation in performance and scores too.

- I assigned max_iter=10 and ran it multiple times in a for loop and performed variance evaluation and picked the best result.
- I also tried with different distance methods : Cosine similarity and euclidean distance
  Cosine similarity is giving better outcomes for this dataset.
  euclidean distance : 0.75
  Cosine similarity: 0.95
- I implemented sum square error and variance evaluation for various k value
  Error decreases as the k values increases
  Variance is increases in this case

```
In [423]: plt.plot(k_val,variances)
Out[423]: [<matplotlib.lines.Line2D at 0x1fea24e3580>]
```




```
In [424]: plt.plot(k_val,error)
Out[424]: [<matplotlib.lines.Line2D at 0x1fea25078e0>]
```



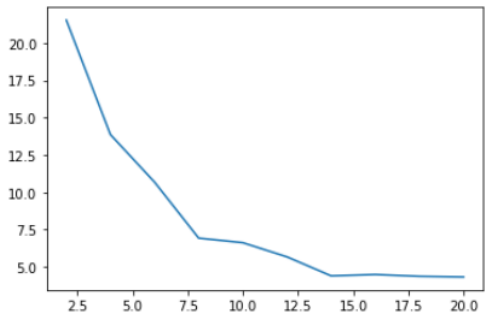
Part-2 : Image Clustering

- In this part we were given an image dataset. Where each image size- 28*28 which are then flattered and represented in 784 features. K=10(given)
- I Used Umap dimensionality reduction on this dataset for better results
  Umap feature reduction is very supportive for clustering algorithms
  It gives various options before clustering

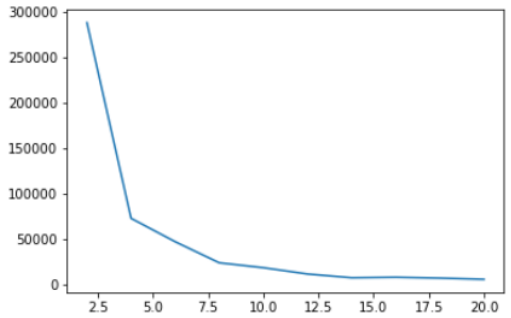
- As parameter I took: n_neighbors=10, min_dist=0.001, n_components=2, metric="euclidean", random_state=42
  Here n_neighbours controls the structure of the data.
  min_dist controls the packing part of UMAP
  metrics computed between points
- data plot after UMAP:

  n_components to reduce the dimensions
- Here Euclidean distance is performing value
- I used variance evaluation the same as above to get the best fit out max_iter runs and sum square error over various K values.
- In this case error decrease as usual where as variance is also decreasing not like above

```
In [302]: plt.plot(k_val,variances)
Out[302]: [<matplotlib.lines.Line2D at 0x1fea0b9af40>]
```




```
In [303]: plt.plot(k_val,error)
Out[303]: [<matplotlib.lines.Line2D at 0x1fea1f28160>]
```



Conclusion:

From the above two cases we can observe that cosine similarity works well for smaller k values as it is based on the angles where euclidean distance works in all cases.
And in part-1 variance is increased before 5 which shows that that dataset can only be clustered in any value below 5. Where as in the case of image dataset it didn't increases which shows that there images can be clustered up to 20 or above as per our observations