# Solution of PiMA Research Section

Phan Chi Tho

June $17^{th}$ 2021

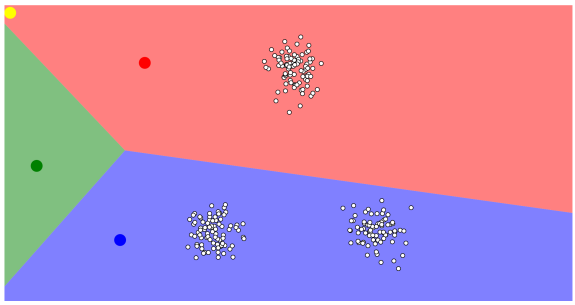
## LT1 Solution:

The K-means clustering algorithm help to group the data set into different clusters which have the same properties. The algorithm inputs are the number of clusters K and the data set. The results of the K-means clustering algorithm are the centroids of the K clusters and labels for the training data (each data point is assigned to a single cluster).
The algorithm tries to minimize the sum of the distances (from the centroid to each data in the cluster of that centroid). The result may be a local optimum, which means that the result may not be the minimum. Therefore, in the same situation with randomized starting centroids may give different outcomes.

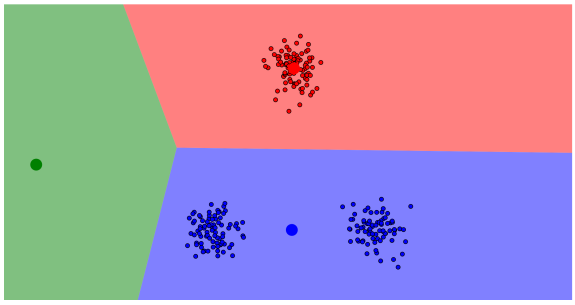
## LT2 Solution:

1. Input the data set.
2. Choose a number of K.
3. Labeling each data points, which is nearest to centroid, into cluster of that centroid.
4. If the position of centroid remains unchanged, stop the algoritm.
5. Taking the mean of all data points assigned to that centroid's cluster to update the position of centroid.
6. Return to step 3.

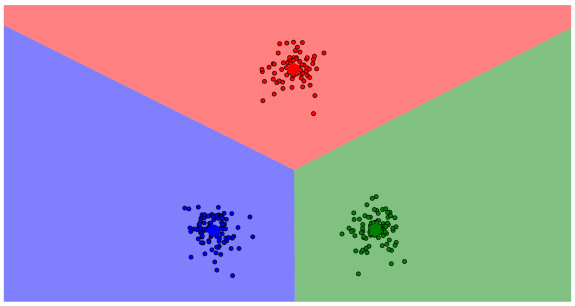
## LT3 Solution:



If we put 2 centroids closer to the data set (in this picture that is the red and blue one) and the other one (green) is further away from the data set compared to the rest. As in the picture, the outcome of the algorithm will not be optimized (as shown in the next picture).



This is because the algorithm will label the data set to the nearest centroid, which is red and blue in the picture mentioned. So that no data will be labeled to the green centroid, after update the position of the centroid, blue and red ones will come closer to the data points while the green one remain the same (no points associated with it). Thus the outcome will not be as expected like in the later picture.



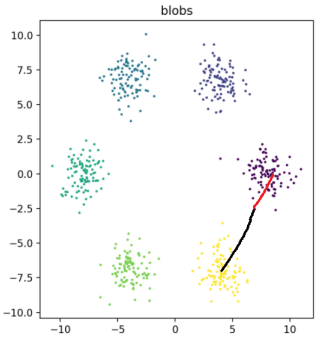
## LT4 Solution:

K-means++ Clustering helps to solve the limitation of the original K-means algorithm, with the input is the amount of K-centroids, and the output is the initial position of those centroids among the data set.
The algorithm can be described as below:

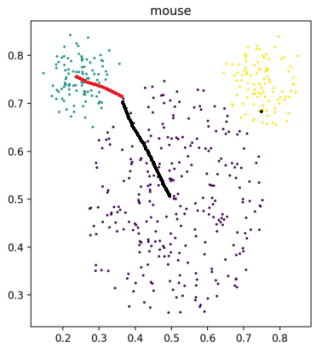
1. Input number of K.
2. Randomly choose among point in dataset for the first centroid.
3. Let $D(x)$ (with $x$ is the data point) be the shortest distance from a data point to its closest centroid selected.
4. Choose point $x_i$ as the new centroid which has maximum probability $\frac{D(x_i)^2}{sum(D(x)^2)}$ (or $D(x_i)$ max).
5. Return to step 2 until we get k centroids.

## LT5 Solution:

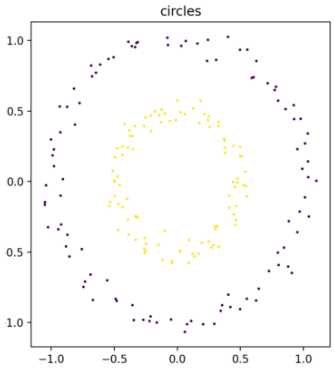
In blobs picture, K means algorithm can give the outcome as expected in the picture. Because the smallest distance of each centroid to data of their counterparts cluster , as in the black line, is still longer than the radius of the counterpart clusters, as in the red line.




However, in mouse datasets, K-means can not apply. The smallest distance, as in red line, centroid (green and yellow) to data of their counterparts cluster (purple) is; however, smaller than the radius of the counterpart clusters, as in black line, the position of that cluster will be closer to that point. So that those data (purple) will eventually change their clusters (into yellow or green). Hence the result will not be as expected.




The K means algorithm also cannot apply to the circle datasets. Because the outcome of K means is separately cluster, while this picture, two clusters are overlapping each other.




## TH2 Solution:

```
44
45  def kmeans_testblobs(n_trials=10, smart=False):
46    data = np.genfromtxt('{}.csv'.format('blobs'), delimiter=',')
47    X, y_true = data[:, :2], data[:, 2]
48    n_clusters = int(np.max(y_true)) + 1
49    scores = np.zeros(n_trials)
50
51    for t in range(n_trials):
52      centroids_mykm, y_pred_mykm = my_kmeans(X, n_clusters=n_clusters, smart=smart)
53      score_mykm = score(y_true, y_pred_mykm)
54      print('Trial t = {}. Score: {}/{}. Accuracy: {}'.format(
55        t+1, score_mykm, y_true.shape[0], score_mykm / y_true.shape[0]
56      ))
57      scores[t] = score_mykm
58
```

`kmeans_testblobs(100, smart=False)`. This means that, in the line 52 in

main.py file, the smart argument will become False. This function then activate the function `my_kmeans` in the `my_kmeans.py`

```
66  def my_kmeans(X, n_clusters, max_iter=100, smart=False):
67      # Khoi tao cac centroid ban dau bang cach chon ngau nhien
68      centroids = (
69          centroids_smart_select(X, n_clusters) if smart
70          else centroids_random_select(X, n_clusters)
71      )
72      labels = - np.ones(X.shape[0])  # khoi tao bang vector [-1, -1, ..., -1]
73      for iteration in range(max_iter):
```

In the line 69-70 we can see that if smart is False, then centroids initial position is randomly chosen. Therefore, the output may not 100% accuracy all the time. On the other hand, `kmeans_testblobs(100, smart=True)`. Similar to above, the line 69-70, when smart argument is True, then centroids initial position will be smart selected. Therefore, the output will be always 100% accuracy.

## TH3 Solution:

As in the LT5 question, the K-means algorithm can not apply to these data set, so that the smart initial position centroid still can not change the result. Applying `centroid_smart_select` algorithm in the code, we can see that the result are relatively the same compared to `centroid_random_select` algorithm. Below are two figure of mouse and circle dataset respectively with smart selected centroid and random selected centroid.

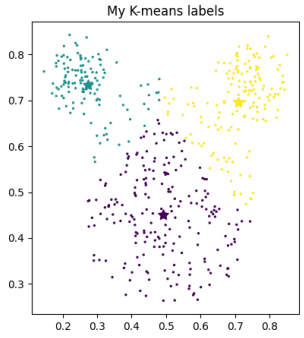



Figure 1: Smart selected centroid

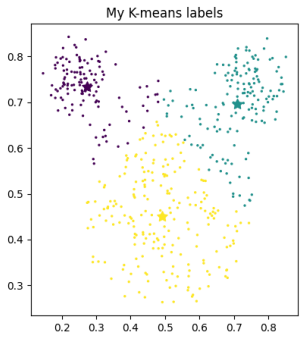



Figure 2: Random selected centroid

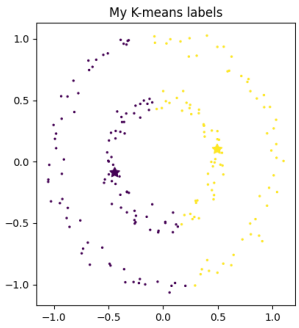



Figure 3: Smart selected centroid

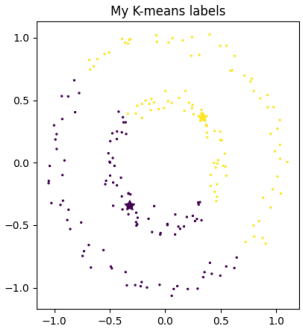



Figure 4: Random selected centroid

## NC4 Solution:

After applying the K-means algorithm in `NC4.py`, K-means can distinguish two clusters with an accuracy of 95% (95 output of algorithm similar to the data, while the remaining are opposite). When changing into three clusters to find the remaining small group, the result will have an accuracy of 89% (89 output is similar to the data, and the remaining will be in the small group).

## NC5 Solution:

The K-means algorithm can't apply in this problem. So that I have created a new clustering version for this situation with `NC5.py` code. The algorithm's input is the id number of the person, corresponding to line in csv file, and its output is all the people which are in the same group to that person. It can demonstrate simply as below:

1. Input the data set.
2. Each mailing data from every person is setting at descending order.
3. Choose a specify person.
4. Find people that he/she mailing most.
5. Find total number of people in that group (the specify person group).
6. Add each person in that group and output the result.

# References

**For searching python syntax:**
https://docs.python.org/3/
**For searching numpy syntax:**
https://numpy.org/doc/
**Using scikit-learn package:**
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
**Definition of K-means Algorithm:**
https://machinelearningcoban.com/2017/01/01/kmeans/
https://blogs.oracle.com/ai-and-datascience/post/introduction-to-k-means-clustering
**Smart-selected centroids Algorithm:**
https://towardsdatascience.com/understanding-k-means-k-means-and-k-medoids-clustering-algorithms-ad9c9fbf47ca#:
http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf