Name: Adel Hamad Alkhamisy

Miner2 username: AA
silhouette_score=0.40
V_Score on miner=0.62
Rank on miner=74

The approach that I used is as follow:

Part 1: develop Kmeans algorithm and test it on IRIS dataset:

Step 1: feature reduction technique: The IRIS dataset is simple and consists of 150 objects and each one has 4 features which are sepal length, sepal width, petal length and petal width in cm. Therefore, it does not need any type of preprocessing. However, I implemented PCA to get 2 components that represent the data to visualize them on two dimension plane and see the effect of my Kmeans algorithm.

Step 2: develop kmean algorithm:

A. Kmean with random initialization technique:

1. I implemented the following Kmean algorithm:

Algorithm 8.1 Basic K-means algorithm.

1: Select K points as initial centroids.
2: **repeat**
3: Form K clusters by assigning each point to its closest centroid.
4: Recompute the centroid of each cluster.
5: **until** Centroids do not change.

- I created a class called KMeans and any instance of this class will have those parameters: **x:dataset**, **k: number of clusters**, **n_init: the number of times the algorithm will run with different centroids, max_iteration: the max number of iterations the algorithm will run in a single run.**
- Create a method called fit_predict()
  this method will create the random centroids and add them in centroids array. Then, I created an array called old_centroids to store old values of centroids of the previous iteration. After that I created a "while loop" to check if the centroids changed by calculating Euclidean distance to whole old and new centroids arrays and iteration<max_iteration. If the values of centroids changed the Euclidean distance will be greater than 0, and the algorithm will generate new centroid for each cluster. In the first iteration, by default, the initial centroids are selected randomly from data points and the old_centroids array are initialized with zeros, therefore, the Euclidean distance will be greater than zero and the "while loop" will run. In the "while loop", I created a clusterPerPoint array to add points to clusters. Then I calculated the Euclidean distance of each point in the dataset and the centroids and store the distance values in distances array, and then I selected the nearest_centroid from distances

array which is the min value in the array and assign the value to clusterPerPoint. Now, each point has its nearest_centroid. After that I copy the values of centroids and store them in old_centroids array. Then, for each cluster, I calculated the average of points that belong to this cluster and update the centroids array accordingly. By run the while loop multiple times until the Euclidean distance of new centroids and old centroids is equal to zero or iteration>max_iteration, the K-means reach convergence. The objective function sum of the squared error (SSE) is used to evaluate quality of a clustering. The best clustering will have the smallest SSE and the worst clustering will have largest SSE.

Strategy minimizing variance within clusters (SSE)

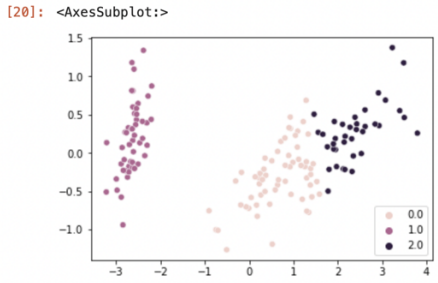
$$\arg min \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mathcal{U}_i||^2$$

B. Kmean with k-means++ technique:

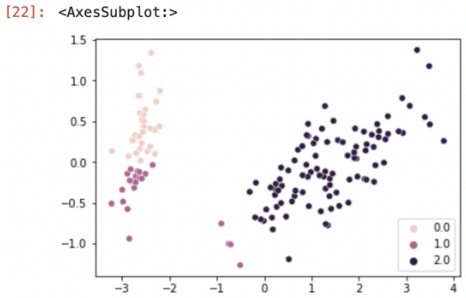
- I created a class called KMeansPlusPlus which is identical to KMeans class but with a change in initial centroids selection.
- In KMeansPlusPlus class, the first initial centroid is picked randomly from data points and append it to the centroid matrix. Then, I created a loop over number of clusters (k). Then, for each data point, I calculated Euclidian distance from chosen centroids and store the min value in distance array. After that I calculated the probability of choosing next centroid by dividing each element of distance array by the sum of all distances. Then, I choose a point as a new centroid with probability proportional to $d(x_i)$

2. Testing Kmean algorithm with random and k-means++ initialization to IRIS dataset:

Random initialization best cluster assignment:



Random initialization worst cluster assignment:



Kmeans++ initialization with best cluster assignment:

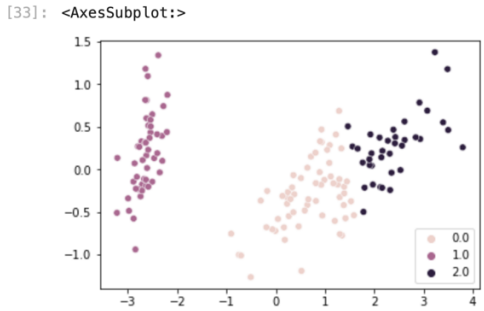


Table 1: Kmean testing on IRIS dataset

Initialization	K (n_clusters)	silhouette_score	V-score
Random (10 iterations in a single run of algorithm)	2	0.68	0.65
	3	0.68	0.73
	4	0.49	0.71
	5	0.37	0.65
	6	0.35	0.64
	7	0.43	0.71
	8	0.26	0.63
	9	0.39	0.61
	10	0.23	0.65
Kmeans++ (10 iterations in a single run of algorithm)	2	0.68	0.65
	3	0.55	0.75
	4	0.49	0.71
	5	0.37	0.65
	6	0.29	0.52
	7	0.34	0.61
	8	0.27	0.61
	9	0.21	0.62
	10	0.20	0.60

Note that Silhouette Coefficient is only defined if number of labels is 2 <= n_labels <= n_samples - 1.

The highest silhouette_score and V_score that I obtained are highlighted with orange. K-means initialization with random centroids obtained 0.68 and 0.73 for silhouette_score and V_score respectively while K-means centroids initialization with k-means++ gave 0.55 and 0.75 for silhouette_score and V_score respectively due to the fact that the k-means++ strategy distributes centroids on the data points better than random initialization. Another notable point

from Table 1 is that the silhouette_score tends to drop while the number of clusters increased and higher values of silhouette_score produces higher value of V_score.

Part 2: testing Kmeans algorithm on MNIST dataset:

Step 1: Preprocessing:

- I imported dataset into numpy array using genfromtxt() function
- I scaled the dataset using StandardScaler to prevent one feature from dominating K-means algorithm. The stander score is calculated using z = (x - u) / s where u is the mean of dataset and s is the standard deviation of dataset according to sklearn documentation.
- Then, I implemented PCA to reduce data dimensionality. However, PCA produced bad silhouette_score and v_score.
- Then, I implemented TSNE to reduce dimensionality which produced a good result with n_components=2 and perplexity=50.0
- After that, I run the K-means algorithm with different parameters values and achieved the highest v_score=0.62 and silhouette_score=0.40 with k-means++ strategy. A notable feature in the Table 2 is that high silhouette_score tends to produce a high v_score in general.

Table 2: Kmean testing on MNIST dataset

Initialization	K (n_clusters)	n_init	max_iteration	PCA	TSNE	silhouette_score	Miner2 Score
Random	10	3	5	3		0.20	0.32
	10	5	8	-	2	0.41	0.55
	10	5	30	-	2	0.42	0.56
	10	10	30	-	2	0.41	0.55
	10	10	30	-	2	0.35	0.60
	10	10	30	-	2	0.35	0.58
	10	10	10	-	2	0.42	0.56
	10	10	10	-	2	0.41	0.55
	10	2	2	-	2	0.36	0.53
Kmeans++	10	3	5	3	2	0.20	0.32
	10	10	30	-	2	0.34	0.60
	10	10	30	-	2	0.33	0.58
	10	5	8	-	2	0.41	0.59
	10	5	8	-	2	0.41	0.56
	10	5	8	-	2	0.41	0.59
	10	5	8	-	2	0.38	0.53
	10	5	8	-	2	0.42	0.55
	10	5	8	-	2	0.41	0.59
	10	5	8	-	2	0.40	0.62
	10	5	8	-	2	0.40	0.55
	10	10	10	-	2	0.40	0.58
	10	5	8	-	2	0.40	0.55
	10	10	10	-	2	0.42	0.57
	2	5	8	-	2	0.41	-
	4	5	8	-	2	0.41	-
	6	5	8	-	2	0.41	-
	8	5	8	-	2	0.41	-
	10	5	8	-	2	0.40	-
	12	5	8	-	2	0.40	-
	14	5	8	-	2	0.40	-
	16	5	8	-	2	0.41	-
	18	5	8	-	2	0.41	-
	20	5	8	-	2	0.41	-