Deniz Aytemiz
denizaytemiz@vt.edu

Programming Assignment 3: K-means Clustering

Part 1: Gaussian Discriminant Analysis

Question 1 :

a) $P(y=1|x) = \dfrac{P(x|y=1).P(y=1)}{P(x)}$

where $P(x) = P(x|y=1).P(y=1) + P(x|y=0).P(y=0)$

$$P(y=1|x) = \frac{P(x|y=1).P(y=1)}{P(x|y=1).P(y=1) + P(x|y=0).P(y=0)}$$

$$= \frac{1}{1 + \dfrac{P(x|y=0).P(y=0)}{P(x|y=1).P(y=1)}}$$

$$\log\left(\frac{P(x|y=0).P(y=0)}{P(x|y=1).P(y=1)}\right) = \log(P(x|y=0)) + \log(P(y=0)) - \log(P(x|y=1)) - \log(P(y=1))$$

$$= \log\left(\frac{1}{(2\pi)^{d/2}.\Sigma^{1/2}}\right) + \log\left(\exp\left(-\tfrac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)\right) + \log(1-\phi)$$

$$- \log\left(\frac{1}{(2\pi)^{d/2}\Sigma^{1/2}}\right) - \log\left(\exp\left(-\tfrac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)\right) - \log(\phi)$$

$$= -\tfrac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) + \log(1-\phi) + \tfrac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) - \log(\phi)$$

$$= \tfrac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) - \tfrac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) + \log\left(\frac{1-\phi}{\phi}\right)$$

(1)

Deniz Aytemiz
denizaytemiz@vt.edu

$$\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) - \frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0) + \log\left(\frac{1-\phi}{\phi}\right) =$$

$$-\theta^T x + \theta_0$$

Then,

$$\theta = \Sigma^{-1}(\mu_1-\mu_0), \quad \theta_0 = -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 + \log\left(\frac{1-\phi}{\phi}\right)$$

b)

$$\log P_\theta(D) = \sum_{i=1}^{n}\log\left(P(x_i|y_i)\right) + \sum_{i=1}^{n}\log\left(P(y_i)\right)$$

$$= \sum_{i=1}^{n}\log\left(\frac{\exp\left(-\frac{(x_i-\mu_{y_i})^2}{2\sigma^2}\right)}{(2\pi)^{d/2}\sigma^2}\right) + \sum_{i=1}^{n}\left(\log\left(\phi^{y_i}(1-\phi)^{1-y_i}\right)\right)$$

⇒ taking derivative w.r.t ϕ, μ_{y_i}, Σ

$$\frac{\partial \log P_\theta(D)}{\partial \phi} = \frac{\partial}{\partial \phi}\sum_{i=1}^{n}\left(y_i\log\phi + (1-y_i)\log(1-\phi)\right)$$

$$= \sum_{i=1}^{n}\left(y_i\cdot\frac{1}{\phi} + \left(-1\cdot\frac{1-y_i}{1-\phi}\right)\right) = \sum_{i=1}^{n}\frac{y_i}{\phi} + \sum_{i=1}^{n}\frac{y_i-1}{1-\phi}$$

$$= \sum_{i=1}^{n}\frac{y_i}{(1-\phi)\cdot\phi} - \sum_{i=1}^{n}\frac{1}{1-\phi}$$

Deniz Aytemiz
denizaytemiz@vt.edu

$$\frac{\partial \log P(D)}{\partial \phi} = \sum_{i=1}^{n} \frac{y_i}{\phi(1-\phi)} - \sum_{i=1}^{n} \frac{1}{1-\phi} = 0 \quad \text{when}$$

$$\sum_{i=1}^{n} \frac{y_i}{\phi(1-\phi)} = \sum_{i=1}^{n} \frac{1}{1-\phi} \Rightarrow \phi = \frac{1}{n} \sum_{i=1}^{n} y_i$$

where $y_i \in \{0,1\}$

hence

$$\phi = \frac{1}{n} \sum_{i=1}^{n} 1\{y_i = 1\}$$

$$\frac{\partial \log P_0(D)}{\partial \mu_0} = \left(\sum_{i=1}^{n} \frac{1}{\sigma^2}(x_i - \mu_{y_i}), \mu_{y_i} \right) \cdot \frac{\partial}{\partial \mu_0} = \frac{1}{\sigma^2} \sum_{i=1}^{n} (1-y_i)(x_i - \mu_0)$$

when $y_i = 0$

$$\frac{\partial \log P_0(D)}{\partial \mu_0} = 0 \quad \text{when} \quad \sum_{i=1}^{n} (1-y_i)(x - \mu_0) = 0$$

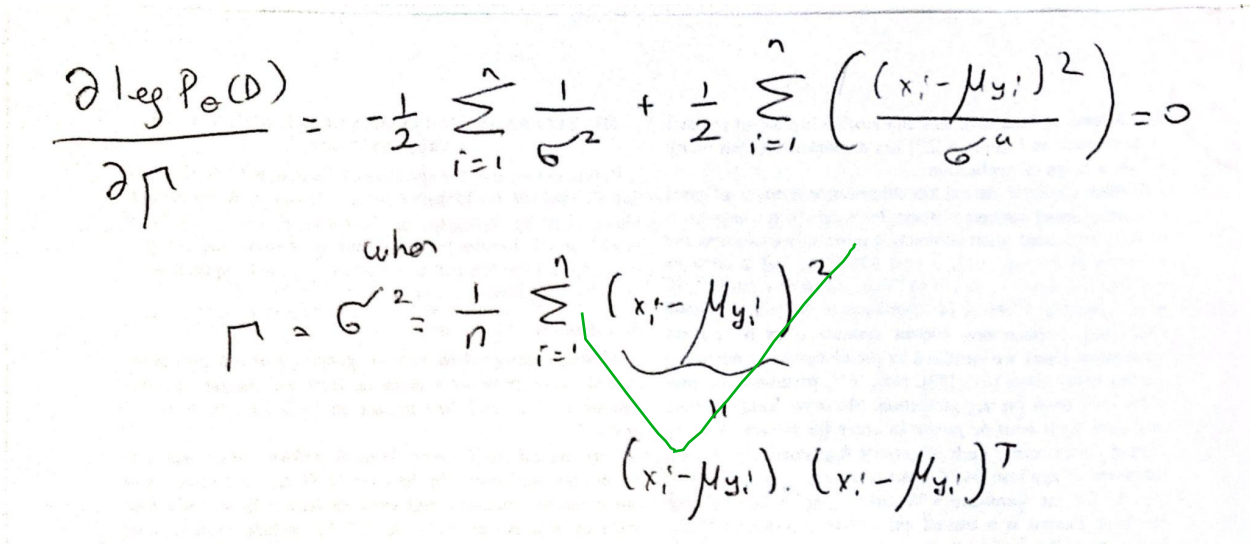
$$\mu_0 = \frac{\sum_{i=1}^{n} 1\{y_i = 0\} x_i}{\sum_{i=1}^{n} 1\{y_i = 0\}}$$

$$\frac{\partial \log P_0(D)}{\partial \mu_1} = \left(\sum_{i=1}^{n} \frac{1}{\sigma^2}(x_i - \mu_{y_i}). \mu_{y_i} \right) \frac{\partial}{\partial \mu_1} \quad \text{when } y_i = 1$$

$$= \frac{1}{\sigma^2} \sum_{i=1}^{n} (x_i - \mu_{y_i}) 1\{y_i = 1\} = 0 \quad \text{when}$$

$$\mu_1 = \frac{\sum_{i=1}^{n} 1\{y_i = 1\} x_i}{\sum_{i=1}^{n} 1\{y_i = 1\} x_i}$$

Deniz Aytemiz
denizaytemiz@vt.edu



Part 2: Coding: K_means clustering

Tasks:

1. Since in k-means clustering we iteratively measure the distance between data and cluster them, we expect all features to contribute same to the distance metric. However for instance if one feature x1 has range [0,1] whereas other feature x2 has range [0,255] the distance between 2 data points with different x1 and x2 values will be affected more by x2 feature's difference since the distances in this feature are wider and hence the effect of feature x1's would be dismissed. In order to avoid that we can scale them both to same range and acquire better representation of data points and distance related to them. This also helps algorithm to convergerge and it makes the results easier to interpret.

2. Code:

```
# k_means_clustering method should return the final cluster centers and
final data labels
def k_means_clustering(X, k, initial_centroids, dist_metric='Euclidian',
max_iter=20):
 '''
 Perform k-means clustering on the input data X.
 Parameters:
   X: input data, numpy array of shape (number of samples, number of
features)
   k: number of clusters
   initial_centroids: a numpy array of shape (k, number of features) with
initial values of the centroids specified by the user
```

Deniz Aytemiz
denizaytemiz@vt.edu

```
   dist_metric: distance metric to use for clustering, default is
'Euclidian'
   max_iter: maximum number of iterations for the algorithm, default is 10
 Returns:
   new_centroids: a numpy array of shape (k, number of features) with
updated centroids
   labels: a numpy array of shape (number of samples, ) with cluster label
ranging from 0 to k-1 for each input data point
 '''

 new_centroids = initial_centroids
 old_centroids = initial_centroids
 labels = []
 for i in range(max_iter):
   labels = []
   #STUDENT CODE HERE:
   # Assign each data point to the closest centroid based on dis_metric
and populate labels for each data points
   for j in range (X.shape[0]):
     dist = old_centroids - X[j]
     norms = np.linalg.norm(dist, axis=1)
     #print(norms)
     cluster_index = np.argmin(norms)
     #print(cluster_index)
     labels.append(cluster_index)
     #print(labels)
   #Update centroids based on new cluster assignments, and store the old
centroids before assigning new values.

   copy_cent = old_centroids.copy()
   #print(copy_cent)
   for l in range(k):
     indices = [m for m, x in enumerate(labels) if x == l]
   #  print(indices)
     if indices:
       new_centroids[l] = np.mean(X[(indices)],axis = 0)# mean by columns,
       print(new_centroids)
     else:
       pass
```

Deniz Aytemiz
denizaytemiz@vt.edu

```
   # Stop if the centroids haven't moved (i.e break if updated centroids
are same as old centroids)
   old_centroids = new_centroids
   #print(new_centroids)
   if np.all(new_centroids == copy_cent):
     break

 return new_centroids, labels
```

4.
Code:

```
def perform_clustering(X, k, dist_metric='Euclidian'):
 '''

  Parameters:
   X: input data, numpy array of shape (number of samples, number of
features)
   k: number of clusters
   dist_metric: distance metric to use for clustering, default is
'Euclidian'

 Returns:
   new_centroids: a numpy array of shape (k, number of features) with
updated centroids
   score: Silhouette score for the current clustering

 '''

 n_samples, n_features = X.shape

 #STUDENT CODE STARTS HERE
 # Randomly initialize the initial centroids (you can use
'np.random.choice' inbuilt function)
 rand_row = np.random.choice(n_samples, size = k, replace = False)
 initial_centroids = X[rand_row]
 #STUDENT CODE ENDS HERE

 #Find the cluster centers using the k_means_clustering function defined
above.
```

Deniz Aytemiz
denizaytemiz@vt.edu

```
 new_centroids, labels = k_means_clustering(X, k,
initial_centroids,dist_metric, 20)

 #Evaluate the quality of the clusters by calculating the Silhouette score
using the cal_sil_score function defined above.
 score = cal_sil_score(X,labels)

  return new_centroids, score
```

Discussion: The algorithm of k-means clustering is sensitive to initial centroids since algorithm firstly clusters the data points according to initial centroids by selecting the cluster which have minimum distance between its centroid and the data point. After first clustering is done, the new centroids are selected according to the first clustering (by taking the mean) and process goes on. Hence if completely different initial centroids are selected, the first clustering of all datapoints can be very different and therefore the rest of the process (the second clustering and rest) may be very different too since the first clustering affects the whole process.
Apart from assigning initial clusters from random points in training data samples as in the code, there is also kmeans++ algorithm. Which selects the first cluster center randomly and then based on its distance to other points, it selects the other cluster centers randomly but this time the probability of selecting another cluster center is inversely proportional to the distance square between the first center. Therefore the probability that cluster centers are spread away from each other is increased.
5.
Code:

```
k_set=[2,3,4,5,6,7,8,9,10] # students would populate
sil_scores = [] # to be populated with corresponding silhouette_scores
#STUDENT CODE HERE:
#Evaluate the Silhouette scores for different k values using the
perform_Clustering method defined above.

for k in k_set:
 new_centroids, score = perform_clustering(X, k, dist_metric='Euclidian')
 sil_scores.append(score)

#Create a plot of the k values versus the Silhouette scores using the
plotting_scores function.
# ploting sil score for different k values
plotting_scores(k_set,sil_scores)
```

Deniz Aytemiz
denizaytemiz@vt.edu

```
#STUDENT CODE ENDS HERE
```

Discussion: The plot of number of cluster centers with respect to silhouette scores is given in Figure 1.

Figure 1

It looks like the best value of k is 4 since it gives a Silhouette score of 0.7 and it is the maximum among other k values. It means when k is 4 the points are closest to their own clusters and farthest away from other clusters which indicates a good k cluster model.

6.

Code:

```
def perform_clustering_new(X, k, dist_metric='Euclidian'):
    k = 5
    n_samples, n_features = X.shape
    # kmeans++ initialization used
    # Select the first centroid randomly from the data points
    centroids = [X[np.random.choice(n_samples)]]

    # Select the remaining k-1 centroids using K-means++ initialization
    for i in range(k-1):
        # Compute the distances of each data point to the nearest centroid
        distances = np.array([min([np.linalg.norm(x-c)**2 for c in
centroids]) for x in X])

        # Choose the data point with the highest distance as the next
centroid
```

```
        next_centroid = X[np.argmax(distances)]
        centroids.append(next_centroid)

    initial_centroids = np.array(centroids)

    #Find the cluster centers using the k_means_clustering function defined
above.
    new_centroids, labels = k_means_clustering(X, k,
initial_centroids,dist_metric, 20)

    #Evaluate the quality of the clusters by calculating the Silhouette
score using the cal_sil_score function defined above.
    score = cal_sil_score(X,labels)

     return new_centroids, score
centroids_kpp, score_pp = perform_clustering_new(X, k=5,
dist_metric='Euclidian')
centroids, score = perform_clustering(X,k=5,dist_metric='Euclidian')
print('score with kmeans++ initialization ', score_pp)
print('score with random initialization ', score)

# Define the initial cluster centroids as a separate array
initial_centroids_pp = centroids_kpp
initial_centroids = centroids

# Create a scatter plot of the scaled data
fig, ax = plt.subplots(figsize=(8,8))
df.plot(ax=ax, kind='scatter', x='x', y='y')

# Plot the initial cluster centroids with random initialization as green
x's
# plot the initial cluster centroids kmeans++ initialization as red points
ax.scatter(initial_centroids_pp[:,0], initial_centroids_pp[:,1],
color='red', marker='o', s=100)
ax.scatter(initial_centroids[:,0], initial_centroids[:,1], color='green',
marker='x', s=100)

# Set the axis labels and show the plot
plt.xlabel('X_1')
plt.ylabel('X_2')
```

```
plt.show()
def perform_clustering_new2(X, k, dist_metric='Euclidian'):
 # pick k*2 clusters at random and eliminate the ones that have the min
distance with other cluster centers,
 #until k cluster centers remain
 n_samples, n_features = X.shape

 #STUDENT CODE STARTS HERE
 # Randomly initialize k*2 initial centroids
 rand_row = np.random.choice(n_samples, size = k*2, replace = False)
 initial_centroids = X[rand_row]
 # with this loop, in each iteration eliminate a centroid that has min
distance from the one indexed as i.
 #so in first iteration, centroid closest to first centroid is eliminated,
in second iteration,
 # centroid closest to second centroid is eliminated and it goes on until
k centroids left
 for i in range(int(len(initial_centroids)/2)):
   dist_centroids = pairwise_distances(initial_centroids)
   min_one_from_first = dist_centroids[i].argsort()
   initial_centroids2 = initial_centroids[0:min_one_from_first[1]]
   initial_centroids3 =
initial_centroids[min_one_from_first[1]+1:len(initial_centroids)]
   initial_centroids =
np.concatenate((initial_centroids2,initial_centroids3),axis=0)

 new_centroids, labels = k_means_clustering(X, k,
initial_centroids,dist_metric, 20)

 #Evaluate the quality of the clusters by calculating the Silhouette score
using the cal_sil_score function defined above.
 score = cal_sil_score(X,labels)

  return new_centroids, score
centroids_2, score_2 = perform_clustering_new2(X, k=5,
dist_metric='Euclidian')
centroids, score = perform_clustering(X,k=5,dist_metric='Euclidian')
print('score with selecting multiple centers and pruning initialization ',
score_2)
print('score with random initialization ', score)
```

Deniz Aytemiz
denizaytemiz@vt.edu

```
# Define the initial cluster centroids as a separate array
initial_centroids_2 = centroids_2
initial_centroids = centroids

# Create a scatter plot of the scaled data
fig, ax = plt.subplots(figsize=(8,8))
df.plot(ax=ax, kind='scatter', x='x', y='y')

# Plot the initial cluster centroids with random initialization as green
x's
# plot the initial cluster centroids with pruning initialization as red
points
ax.scatter(initial_centroids_2[:,0], initial_centroids_2[:,1],
color='red', marker='o', s=100)
ax.scatter(initial_centroids[:,0], initial_centroids[:,1], color='green',
marker='x', s=100)

# Set the axis labels and show the plot
plt.xlabel('X_1')
plt.ylabel('X_2')
plt.show()
```

Discussion:
For initialization of the cluster centers I have tried kmeans++ in `perform_clustering_new` function. It selects the first cluster center at random from data samples. Then it computes the distances of all data samples from that center and chooses the data sample that has the most distance from the center as the next cluster center. It repeats this process until ‘k’ centers are selected. This algorithms increases chances of the center points to be as much as spread out as possible, therefore expected in less overlapping clusters.
The silhouette score for k = 5, with kmeans++ initialization is 0.7474285056583688,
The silhouette score for k = 5, with random initialization is 0.6056858582684054,
In figure 2 below, the converged cluster centers with both initialization methods for k is 5 is given. Green x’s indicate the centers found by random initialization and red dots indicate centers found by kmeans++ initialization. As it can be observed, some of the centers are overlapping where some centers are very different from each other since random initialization can select any 5 points, no matter what the distance is and if some initial centers are too close or too far from each other the resulting convergence may be as follows. Overall initialization with kmeans++ looks better. I should also indicate that, after multiple reruns of the code, some outputs I have got was as in Figure 2(hence also gave different silhouette scores) whereas for the rest, the centers were completely overlapping and converging to same cluster centers (hence giving the same

silhouette scores). That is due to the random initialization's selection of initial centers, in some runs it selects more evely distributed centers and we obtain a better silhouette score. To indicate the effects of different initialization methods, in this report, I have included the plot where the cluster centers and silhouette scores resulted differently as in Figure 2.

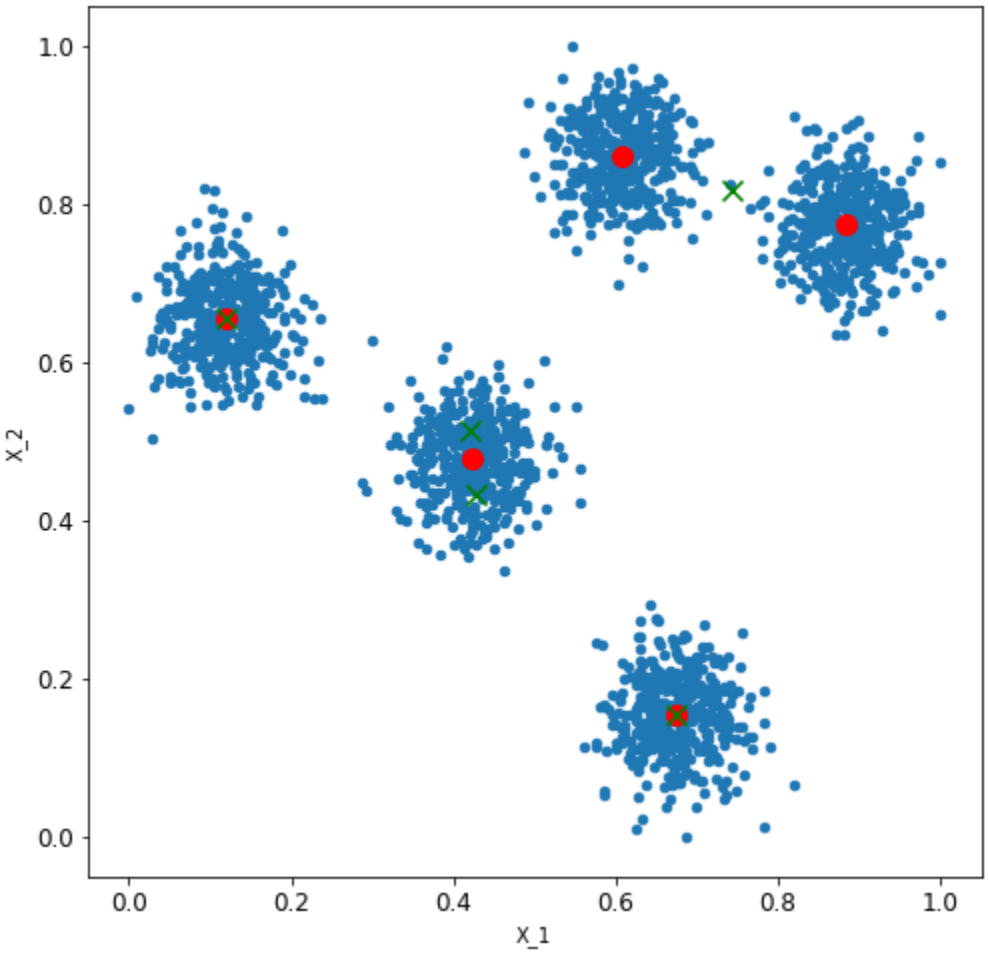


Figure 2

Apart from the kmeans++ initialization method, I have also tried selecting k*2 initial centers and pruning them with based on their distance from each other. The resulting converged centers are shown in Figure 3.

Deniz Aytemiz
denizaytemiz@vt.edu

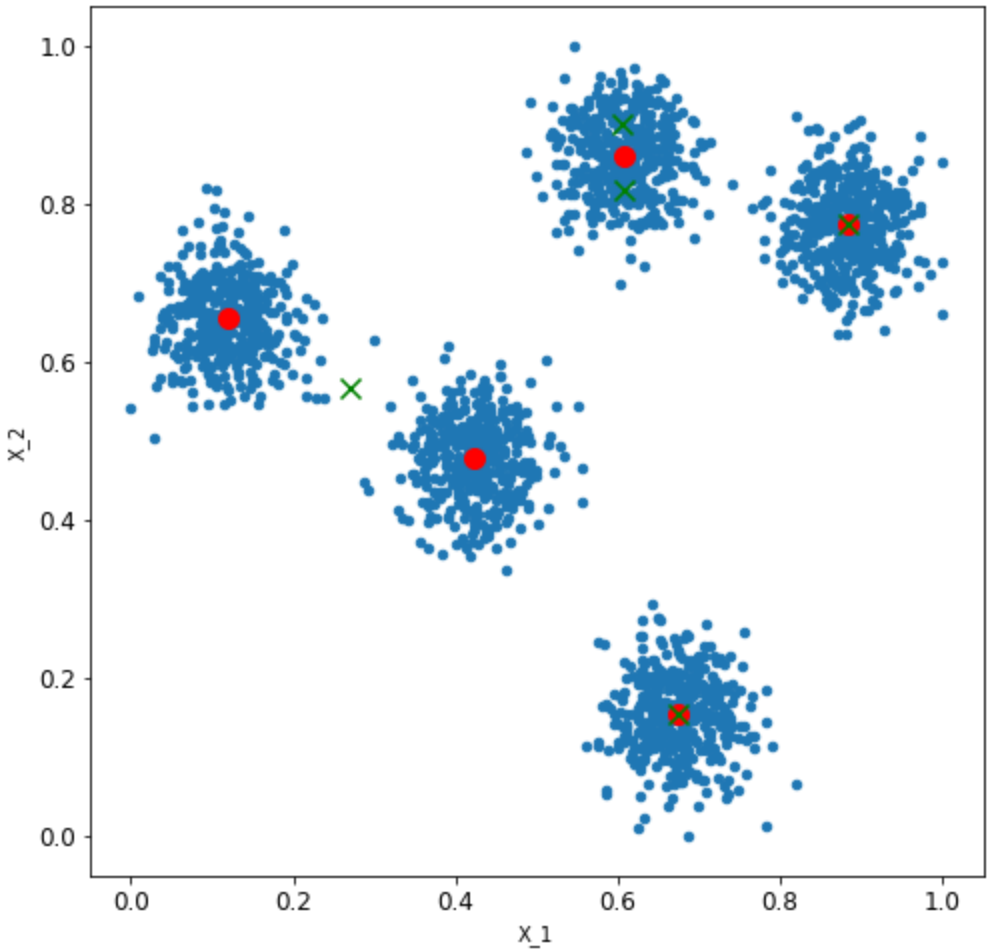


Figure 3

The red dots indicate the converged centers used pruning-method initialization whereas the green crosses indicate the converged centers used random initialization. The silhouette scores for both are; score with selecting multiple centers and pruning initialization 0.7474285056583688, score with random initialization 0.5728770240756165. The pruning algorithm works by intitializing k*2 centers at randomly and starting from the first center, one by one it eliminates the centers with min distance to the center under consideration. The elimination continues until there are k centers left.

7.

Code:

```
sil_scores = []
k_set = np.arange(2,11).astype(int)

X = df.values.copy()
#STUDENT CODE STARTS HERE:
# Feature scaling: Scale the data using minmax scaler
X = sc.fit_transform(X)
```

Deniz Aytemiz
denizaytemiz@vt.edu

```
for k in k_set:
 new_centroids,score = perform_clustering(X,k,dist_metric = 'Euclidian')
 sil_scores.append(score)
 plotting_scores(k_set,sil_scores)
#STUDENT CODE ENDS HERE
```

Discussion:
In Figure 4 the number of clusters vs the sil_score is plotted for the rea-world dataset.

Figure 4

The best k value according to the plot above is 2, which gives the max silhouette score of just above 0.18, which is not as good as the results obtained with the generated dataset but it may be due to this real world dataset's samples are much more overlapped and they can not be grouped apart from each other as much as the generated dataset.

8.
Code:

```
df = pd.read_csv('household_power_consumption_hourly.csv')
kmeans = KMeans(n_clusters=3)
cluster_found = kmeans.fit_predict(X)
cluster_found_sr = pd.Series(cluster_found, name='cluster')
df = df.set_index(cluster_found_sr, append=True )

fig, ax= plt.subplots(1,1, figsize=(18,10))
color_list = ['blue','red','green']
cluster_values = sorted(df.index.get_level_values('cluster').unique())

for cluster, color in zip(cluster_values, color_list):
```

Deniz Aytemiz
denizaytemiz@vt.edu

```
    df.xs(cluster, level=1).T.plot(
        ax=ax, legend=False, alpha=0.01, color=color, label= f'Cluster
{cluster}'
        )
    df.xs(cluster, level=1).median().plot(
        ax=ax, color=color, alpha=0.9, ls='--'
    )

ax.set_xticks(range(0, 24), range(0, 24))
ax.set_ylabel('kilowatts')
ax.set_xlabel('hour')
# ax.legend()
clusterer = KMeans(n_clusters=2, n_init="auto", random_state=10)
cluster_labels = clusterer.fit_predict(X)
silhouette_avg = silhouette_score(X, cluster_labels)

silhouette_avg
```

Discussion:
For k =2,

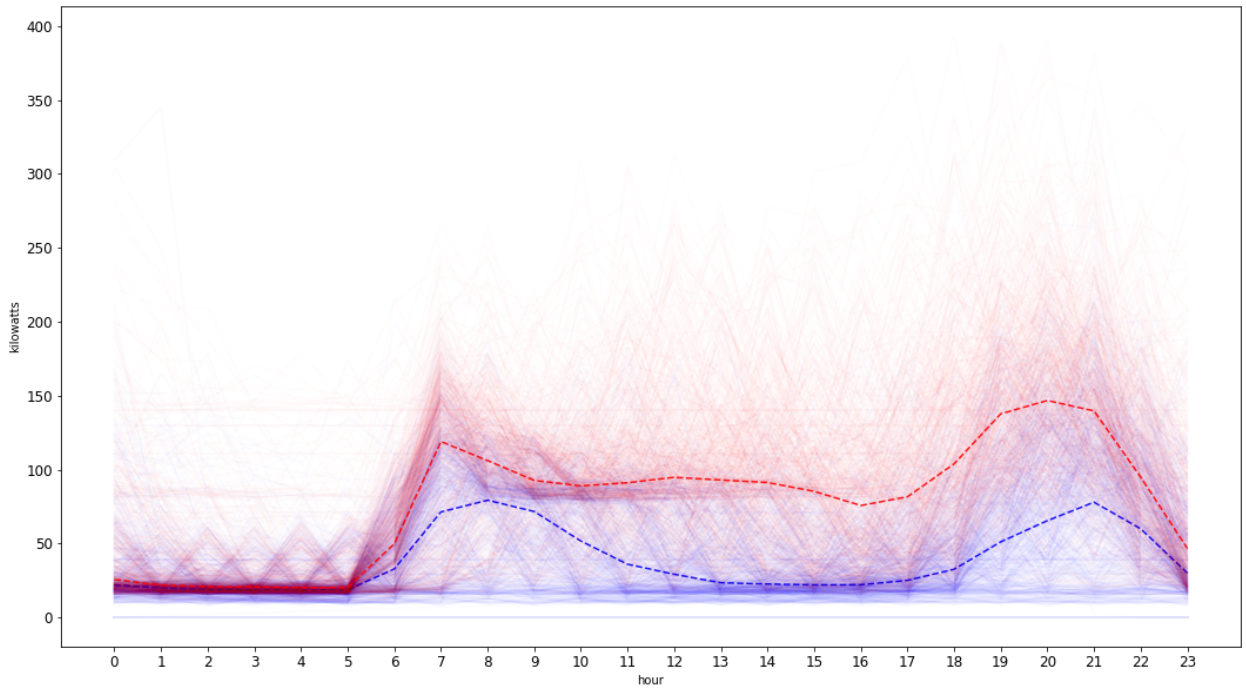


Figure 5: k-means clustering for k = 2 on household_power_consumption dataset

Deniz Aytemiz
denizaytemiz@vt.edu

In Figure 5, the dashed lines indicating center clusters, make sense since they seem to be passing from the average values(where the most overlap of same colored data occurs) of red and blue indicating clusters. It looks like a good indicator for the data split into 2 groups.

For k = 3,

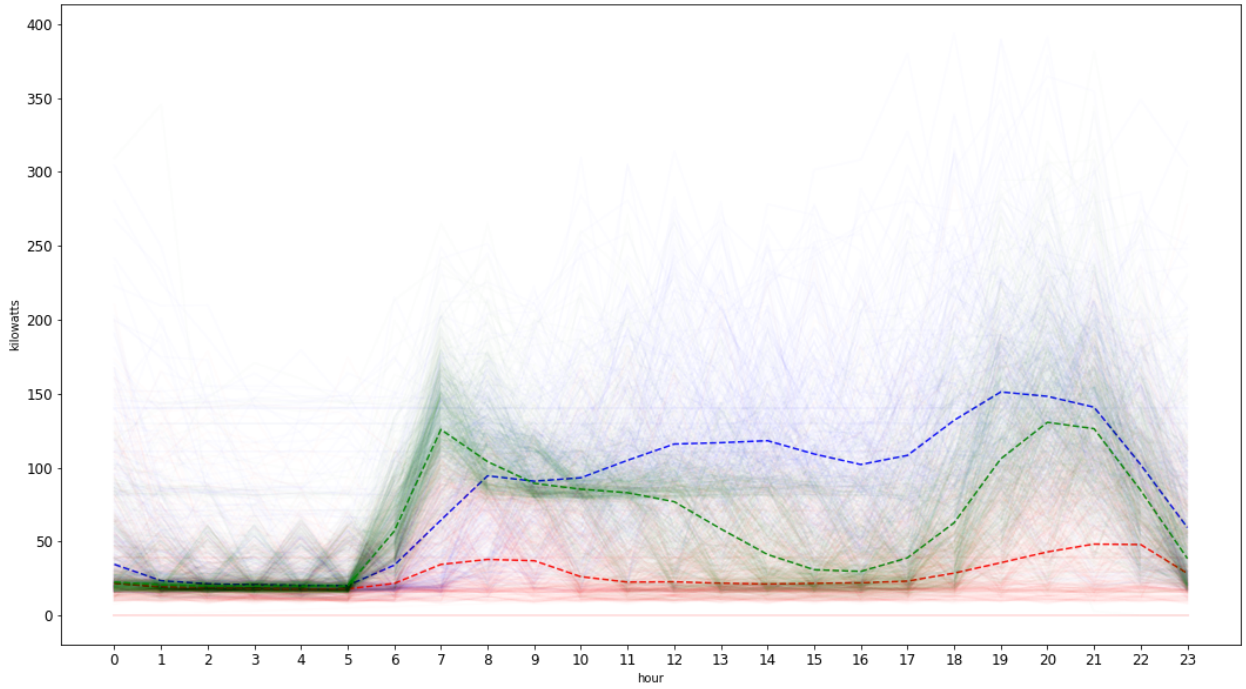


Figure 6: k-means clustering for k = 3 on household_power_consumption dataset

The plot in Figure 6 is still a good indicator for data splitted into 3 groups, as the cluster centers follows the most observable, denser color patterns in the plot. It also can be observed that as the cluster number increases, the overlapping of data in certain hour ranges become more observable. For example in first 5 hours almost all data points have very similar kilowatts values, also some overlapping between hours 8-10 can be observed. Which makes it easier to interpret the silhouette scores being lower than the previous dataset.

To see what happens I increased to big number of clusters, I add k = 5's plot in the below plot

Deniz Aytemiz
denizaytemiz@vt.edu

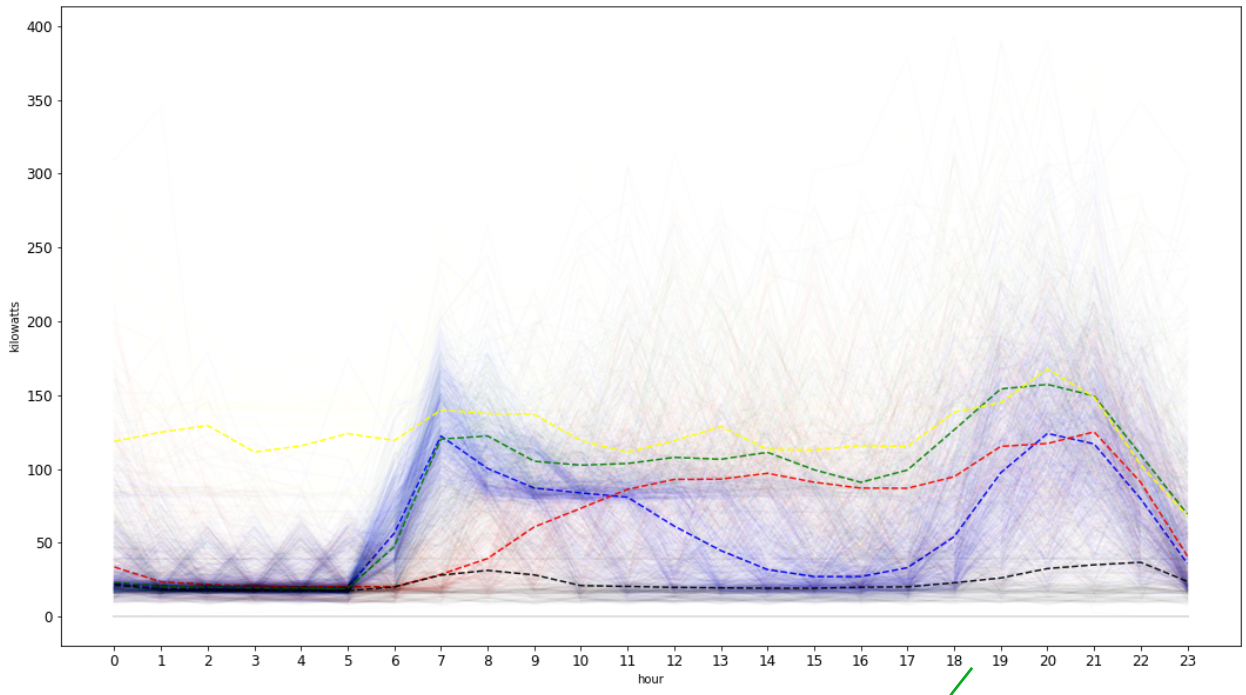


Figure 7: k-means clustering for k = 5 on household_power_consumption dataset

In the Figure 7, again there is some overlap in certain hours and the centers are not very much spread out from each other, however even though k =2 gives the best result, there is no dramatic difference between data clustering of k =2 and k =5. One notable difference is that the yellow cluster can be seen identifying the data samples(the days) that have larger values then 100 kilowatts in the first 5 hours.