Lab 12

Jennifer Lin jenniferyjlin@berkeley.edu

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1

Perform another type of clustering of your choosing on the lab 12 data 01.xlsx dataset and explain which type it is.

```
from google.colab import files
uploaded = files.upload()
# select the file from local drive
```

Choose Files lab_12_data_01.xlsx

- **lab_12_data_01.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 111880 bytes, last modified: 4/24/2020 - 100% done

```
Saving lab_12_data_01.xlsx to lab_12_data_01.xlsx
```

```
import io
df1 = pd.read_excel(io.BytesIO(uploaded['lab_12_data_01.xlsx']))
# df1 = pd.read_excel('lab_12_data_01.xlsx')
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_values = scaler.fit_transform(df1)
df1.loc[:,:] = scaled_values
```

Mini Batch K-means

Mini batch k-means works similarly to the k-means algorithm except that in mini batch k-means the most computationally costly step is conducted on only a random sample of observations as opposed to all observations. This approach can significantly reduce the time required for the algorithm to find convergence (i.e. fit the data) with only a small cost in quality. Mini Batch K-means uses small random batches of data of a fixed size, so they can be stored in memory. Each iteration a new random sample from the dataset is obtained and used to update the clusters and this is repeated until convergence.

```
from sklearn.cluster import MiniBatchKMeans
MBKM = MiniBatchKMeans(n_clusters=16).fit(df1)
centers_MBKM = MBKM.cluster_centers_
print(centers_MBKM)
cluster_labels_MBKM = MBKM.labels_
print(cluster_labels_MBKM)
```

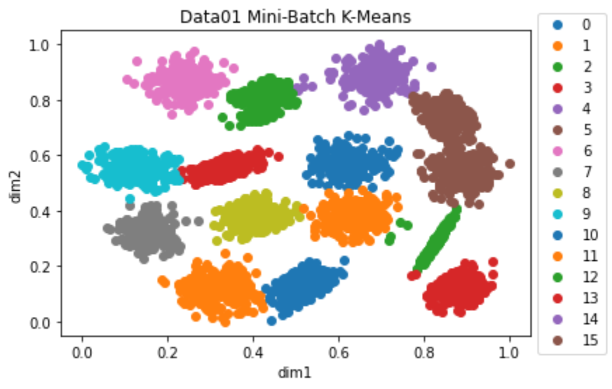
```
[[0.62613156 0.57246498]
 [0.32413464 0.12180752]
 [0.83044467 0.29234342]
 [0.3351529  0.55175766]
 [0.57568717 0.86672222]
 [0.88731868 0.54549524]
 [0.24313268 0.86919429]
 [0.15869122 0.32128681]
 [0.40038619 0.38368175]
 [0.1288095  0.550128  ]
 [0.51330998 0.13673704]
 [0.63289715 0.37621297]
 [0.42469937 0.79955486]
 [0.88483519 0.11804726]
 [0.69495102 0.88501865]
 [0.8498031  0.74173164]]
[ 0  0  0 ... 14  4 14]
```

```
df1_MBKM = df1.copy()
df1_MBKM["cluster"] = cluster_labels_MBKM
df1_MBKM.head()
```

	dim1	dim2	cluster
0	0.683912	0.543504	0
1	0.685701	0.551136	0
2	0.612810	0.570245	0
3	0.635553	0.544047	0
4	0.653693	0.605594	0

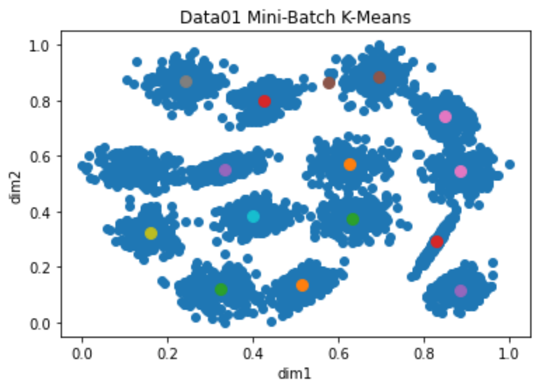
```
groups = df1_MBKM.groupby("cluster")
for name, group in groups:
    plt.plot(group["dim1"], group["dim2"], marker="o", linestyle="", label=name)
plt.title('Data01 Mini-Batch K-Means')
plt.xlabel('dim1')
plt.ylabel('dim2')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

```
<matplotlib.legend.Legend at 0x7f64b94b1358>
```



```
plt.close()
plt.scatter(df1['dim1'],df1['dim2'])
for center in centers_MBKM:
    plt.scatter(center[0],center[1],s = 64)
plt.title('Data01 Mini-Batch K-Means')
plt.xlabel('dim1')
plt.ylabel('dim2')
```

```
Text(0, 0.5, 'dim2')
```



2

From the KMeans clustering demo in lab section, find the variance within each cluster. You will have to call upon the labels attribute instead of cluster centers to assign a cluster index to each row in the data frame and then use the .iloc to extract the desired rows for calculation of the variance.

```
from sklearn.cluster import KMeans
KM = KMeans(n_clusters = 16, random_state = 0).fit(df1)
centers_KM = KM.cluster_centers_
print(centers_KM)
cluster_labels_KM = KM.labels_
print(cluster_labels_KM)
```

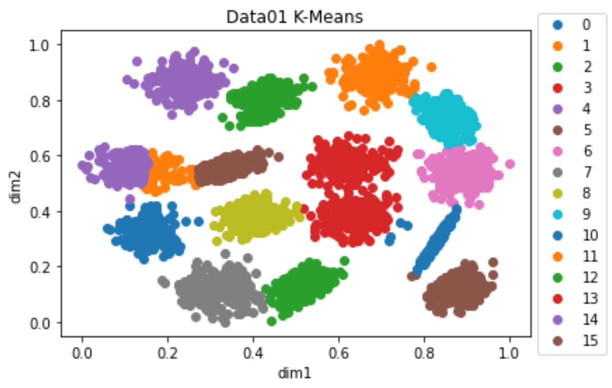
```
[[0.15711562 0.32261899]
 [0.6913367  0.88238   ]
 [0.51796521 0.13536829]
 [0.63449396 0.3788277 ]
 [0.23863291 0.86612737]
 [0.88335561 0.11587698]
 [0.89066842 0.5384078 ]
 [0.31924683 0.12004855]
 [0.40198972 0.38464616]
 [0.85320246 0.73921401]
 [0.82981478 0.29359729]
 [0.18612489 0.54140646]
 [0.42241581 0.80018811]
 [0.6227895  0.56906726]
 [0.1118469  0.55371528]
 [0.33955199 0.55657383]]
[13 13 13 ...  1  1  1]
```

```
df1_KM = df1.copy()
df1_KM["cluster"] = cluster_labels_KM
df1_KM.head()
```

	dim1	dim2	cluster
0	0.683912	0.543504	13
1	0.685701	0.551136	13
2	0.612810	0.570245	13
3	0.635553	0.544047	13
4	0.653693	0.605594	13

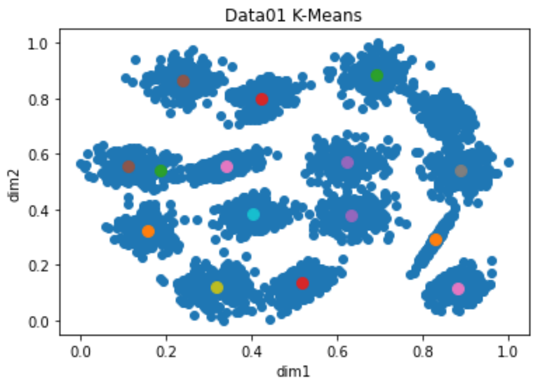
```
groups = df1_KM.groupby("cluster")
for name, group in groups:
    plt.plot(group["dim1"], group["dim2"], marker="o", linestyle="", label=name)
plt.title('Data01 K-Means')
plt.xlabel('dim1')
plt.ylabel('dim2')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

```
<matplotlib.legend.Legend at 0x7f64b94b1400>
```



```
plt.close()
plt.scatter(df1['dim1'],df1['dim2'])
for center in centers_KM:
    plt.scatter(center[0],center[1],s = 64)
plt.title('Data01 K-Means')
plt.xlabel('dim1')
plt.ylabel('dim2')
```

```
Text(0, 0.5, 'dim2')
```



```
from scipy.spatial import distance
df1_KM_variance = []
for i in range(16):
  variance_i = 0
  df_cluster_i = df1_KM.loc[df1_KM['cluster'] == i,].reset_index(drop=True)
  center_i = centers_KM[i,]
  df_cluster_i_pointNum = len(df_cluster_i["dim1"])
  for j in range(df_cluster_i_pointNum):
    a = [ df_cluster_i.loc[j,'dim1'], df_cluster_i.loc[j,'dim2'] ]
    dst_i_j = distance.euclidean(a, center_i)
    variance_i += dst_i_j**2
  df1_KM_variance.append(variance_i)
```

```
[0.596432361533961,
 0.824825781099775,
 0.6464685219267605,
 0.7759431078624995,
 0.7689389592668755,
 0.5454609006280192,
 0.7864277593652275,
 0.9847650213841543,
 0.6428667431843527,
 0.5863350924057685,
 0.4159030260432644,
 0.17568429973663735,
 0.6348880286924055,
 0.7181808870989528,
 0.42645712217700765,
 0.4601553291751772]
```

```
d = {'cluster': range(16), 'variance': df1_KM_variance}
variance_result = pd.DataFrame(data=d)
variance_result
```

3

Read in the file lab 12 data 02.xlsx and make a new Data.Frame out of the columns Annual Income (k$) & Spending Score (1-100).

```
# from google.colab import files
uploaded = files.upload()
# select the file from local drive
```

```
Choose Files  lab_12_data_02.xlsx
```

- **lab_12_data_02.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 15504 bytes, last modified: 4/24/2020 - 100% done

```
Saving lab_12_data_02.xlsx to lab_12_data_02.xlsx
```

```
# import io
df2 = pd.read_excel(io.BytesIO(uploaded['lab_12_data_02.xlsx']))
# df2 = pd.read_excel('lab_12_data_02.xlsx')
```

```
    10        10    0.415903
```

```
df2_income_spending = df2[["Annual Income (k$)","Spending Score (1-100)"]]
df2_income_spending.head()
```

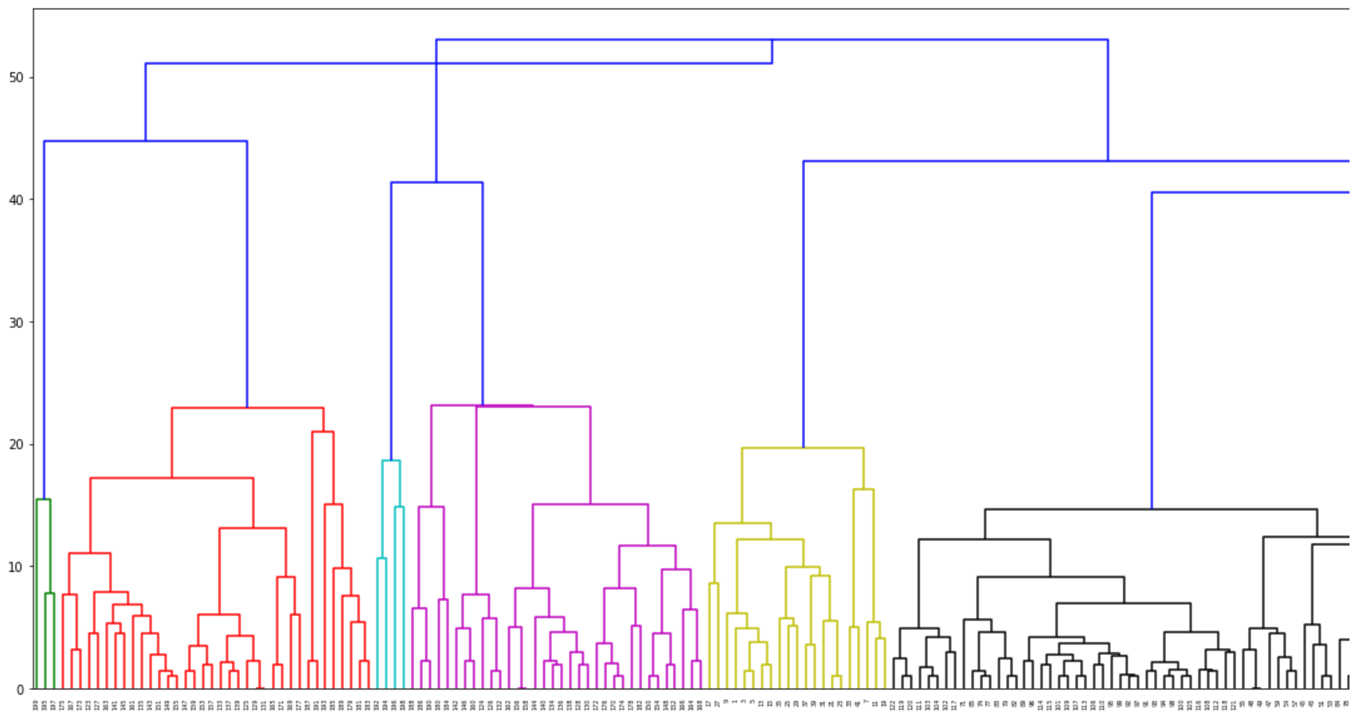
```
     Annual Income (k$)  Spending Score (1-100)
0                    15                      39
1                    15                      81
2                    16                       6
3                    16                      77
4                    17                      40
```

3 (a)

Perform centroid hierarchical clustering on the dataset and visualize it in a dendogram.

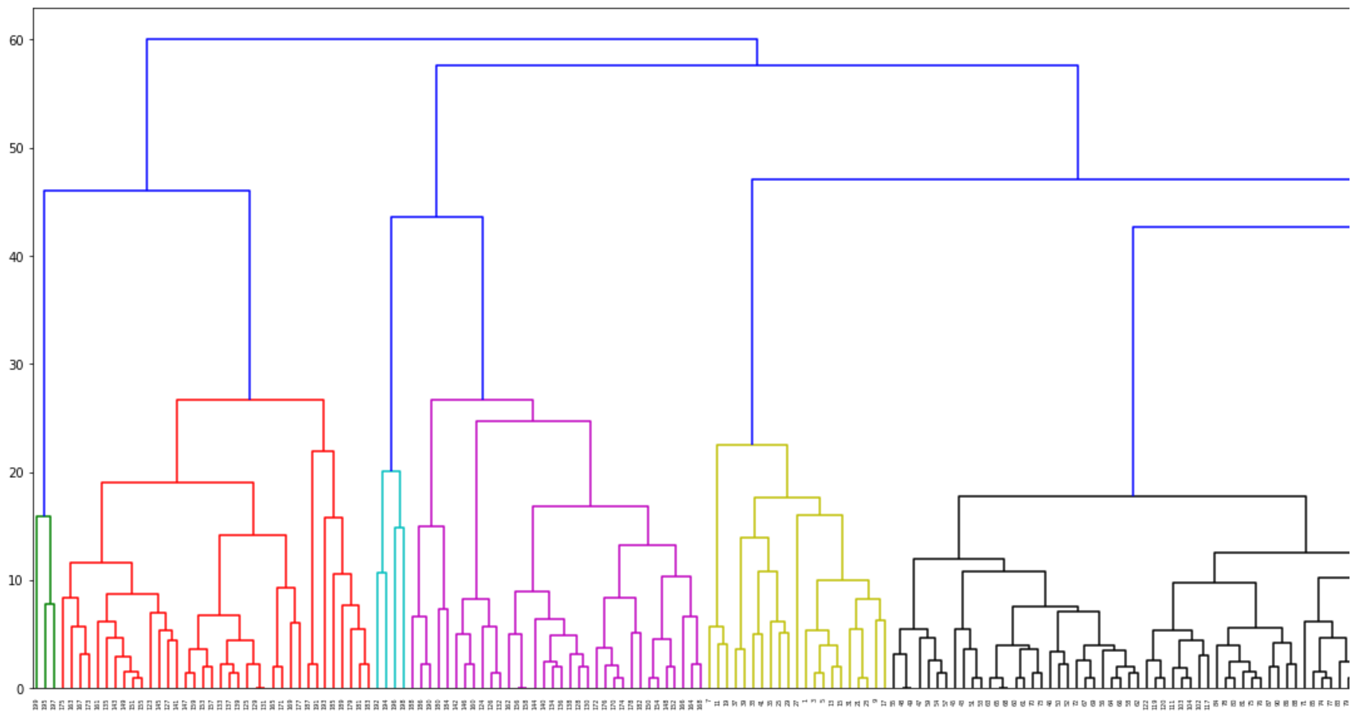
```
# Use the linkage function from scipy.cluster.hierarchy to create the clusters dendogram function in sklearn to visuali
from scipy.cluster.hierarchy import linkage, dendrogram
```

```
linkage_centroid = linkage(df2_income_spending, 'centroid')
fig_centroid = plt.figure(figsize=(25, 10))
dn_centroid = dendrogram(linkage_centroid)
```



3 (b)

```
linkage_UPGMA = linkage(df2_income_spending, 'average')
fig_UPGMA = plt.figure(figsize=(25, 10))
dn_UPGMA = dendrogram(linkage_UPGMA)
```



4

From the lecture notes, please explain in your own words what the curse of dimensionality means and what could be done to overcome it whe analyzing high-dimensional data.

> What is curse of dimensionality?
>
> > The curse of dimensionality refers to the problems arise when analyzing high-dimensional data (e.g. hundreds or thousands of dimensions) that do not occur when analyzing low-dimensional data. For example, when the dimension increases, the distance between different data points become similar to each other. It may causes problems when we try to directly cluster the high-dimensional data.

> How to overcome it?
>
> > Dimensionality reduction is an important technique to overcome the curse of dimensionality. Dimensionality reduction converts the high-dimensional data into low-dimensional data without losing the essensial information of the data. For example, we can use PCA to keep the principal components of the data in order to reduce the dimensionality and prevent the curse of dimensionality.

5

Name a few types of distance metrics used in clustering algorithms.

> Euclidean Distance
>
> > Euclidean distance is the ordinary straight-line distance between two points in Euclidean space.

> Angular/Cosine Distance
>
> > Angular/Cosine Distance calculates the normalized angle between the two vectors. This Angular/Cosine distance metric can then be used to compute a similarity function bounded between 0 and 1. Thus, it is possible that two

> > points have large Euclidean distance and small Angular/Cosine Distance at the same time because they are on the same (or similar angular) vector.
>
> Hamming Distance
>
> > Hamming distance is used to compute the distance between two strings. For instance, we can calculate the distance between two binary data strings. Or more often in biological fields, we can use Hamming distance to

6

Did we employ bottom-up or top-down hierarchical clustering methods in this lab?

> Agglomerative (bottom-up)
>
> > In agglomerative (bottom-up clustering) method, each data point starts in its own cluster. Then, we compute the similarity (e.g., distance) between each of the clusters and join the two most similar clusters. Through this process, pairs of clusters are merged as one moves up the hierarchy. Eventually, there is only a single cluster left. Thus, when we use agglomerative (bottom-up clustering) method, we need to decide when to stop when clustering.

> Divisive (top-down)
>
> > In divisive (top-down) approach, at first, all the data points are treated as one big cluster. Then, we divide the one big cluster into several small clusters. In this process, we split the clusters recursively as one moves down the hierarchy. Finally, there is one cluster for each data point.

> In this lab, we mainly use the agglomerative (bottom-up clustering) method.