Bill Yerkes

CS5542 Big Data Apps and Analytics

In Class Programming –7
8th October 2020

Submit ICP Feedback in Class. : Lnik to Feed back Form

K-means clustering:

Use a different data and use the model provided in ICP7 to perform clustering. You must try 5 different number of clusters (for example n_clusters= 5 or n_clusters=6,7,8,or 9 etc) based on elbow curve and for each cluster visualize the clustering results and report your findings in detail.

ICP Requirements:

1) Successfully executing the code and trying 5 different number of clusters based on elbow curve and visualizing those clusters using python plotting libraries (75 points)
2) Using a new and good dataset (5 points)
3) overall code quality (10 points)
4) Pdf Report quality, video explanation (10 points)

Submission Guidelines:

Same as previous ICPs.

ICP Report:

What I learned in the ICP:

I learned about data clustering. I was able to put into practice what we went over in the lecture on Tuesday. I practice with different data sets, and settled on a data set of latitude and longitude information about craiglist car sales. I also saw that when the data is connected it is hard to see how the different clusters will form, but after they are formed you can see how the groups are related and that there is separation in the data that is not initially visible.

Description of what task I was performing:

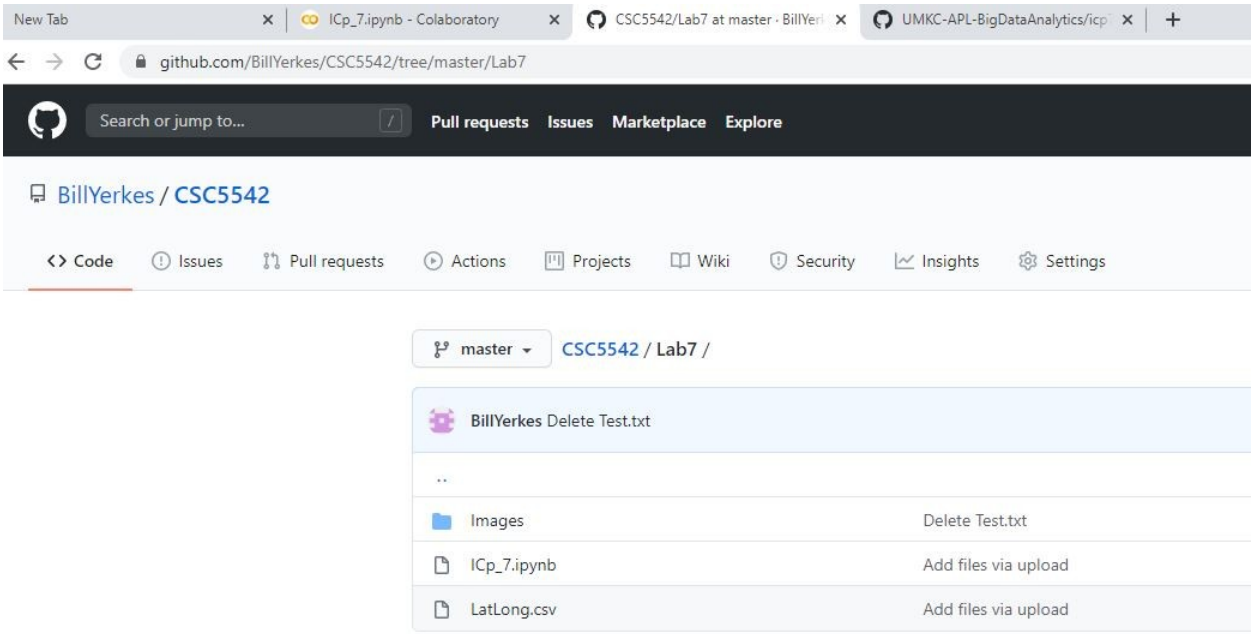
Process new data set, normalize the data, and find the “elbow” to determine the optimal number of clusters.

Challenges I faced:

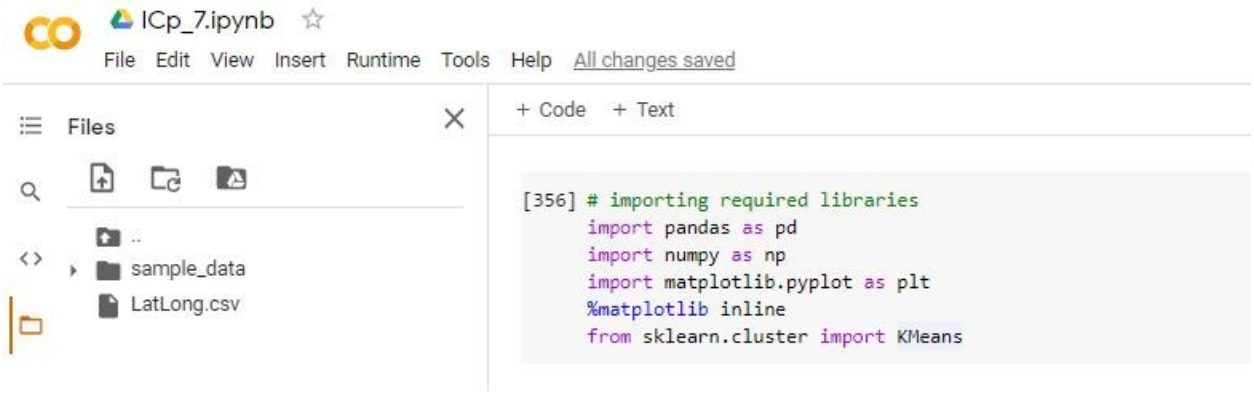
Finding a data set to work with. Cleaning the data and removing rows which caused the different steps to fail.

Screen Shots

GitHub:



Initialize and Install Libraries



Load Data, Describe Data

Data of Car Sales from Craiglist, filterd down to the latatude and longitude for the advertisement.

```
[357] # reading the data and looking at the first five rows of the data
      #data=pd.read_csv("Wholesale customers data.csv")
      data=pd.read_csv("LatLong.csv")
      data.head()
```

	lat	long
0	29.6577	-81.6595
1	31.0710	-97.3898
2	31.0710	-97.3898
3	40.1827	-81.0451
4	40.4845	-81.4358

The aim of this problem is to segment the clients of a wholesale distributor based on their annual spending on diverse product categories, like milk, grocery, region, etc. So, let's start coding!

We have the spending details of customers on different products like Milk, Grocery, Frozen, Detergents, etc. Now, we have to segment the customers based on the provided details. Before doing that, let's pull out some statistics related to the data:

```
[358] # statistics of the data
      data.describe()
```

	lat	long
count	39352.000000	39352.000000
mean	36.885930	-91.447833
std	5.722752	16.342274
min	25.655600	-123.823000
25%	33.245200	-97.866600
50%	37.259300	-84.411800
75%	41.637300	-80.215300
max	46.234800	-67.407300

Standardize the Data

Here, we see that there is a lot of variation in the magnitude of the data. Variables like Channel and Region have low magnitude whereas variables like Fresh, Milk, Grocery, etc. have a higher magnitude.

Since K-Means is a distance-based algorithm, this difference of magnitude can create a problem. So let's first bring all the variables to the same magnitude:

```
[359] # standardizing the data
      from sklearn.preprocessing import StandardScaler
      scaler = StandardScaler()
      data_scaled = scaler.fit_transform(data)

      # statistics of scaled data
      data_s=pd.DataFrame(data_scaled).describe()
      data_s
```

	0	1
count	3.935200e+04	3.935200e+04
mean	-1.979093e-15	8.624513e-16
std	1.000013e+00	1.000013e+00
min	-1.962425e+00	-1.981094e+00
25%	-6.361932e-01	-3.927757e-01
50%	6.524394e-02	4.305474e-01
75%	8.302702e-01	6.873386e-01
max	1.633653e+00	1.471083e+00

The magnitude looks similar now. Next, let's create a kmeans function and fit it on the data:

```
[360]
      # defining the kmeans function with initialization as k-means++
      #kmeans = KMeans(n_clusters=2, init='k-means++')
      kmeans = KMeans(n_clusters=5, init='k-means++')

      # fitting the k means algorithm on scaled data
      kmeans.fit(data_scaled)
```

Inertia on the Fitted Data

We have initialized two clusters and pay attention – the initialization is not random here. We have used the k-means++ initialization which generally produces better results.

Remember in lecture we said that this algorithm randomly initialize the centroids in k-means clustering? Well, this is also potentially problematic because we might get different clusters every time. So, to solve this problem of random initialization, there is an algorithm called K-Means++ that can be used to choose the initial values, or the initial cluster centroids, for K-Means.

In some cases, if the initialization of clusters is not appropriate, K-Means can result in arbitrarily bad clusters. This is where K-Means++ helps. It specifies a procedure to initialize the cluster centers before moving forward with the standard k-means clustering algorithm.

Using the K-Means++ algorithm, we optimize the step where we randomly pick the cluster centroid. We are more likely to find a solution that is competitive to the optimal K-Means solution while using the K-Means++ initialization.

Let's evaluate how well the formed clusters are. To do that, we will calculate the inertia of the clusters:

```
[361] # inertia on the fitted data
      kmeans.inertia_
```

```
9800.46197549601
```

Elbow Curve

how can we decide the optimum number of clusters? One thing we can do is plot a graph, also known as an elbow curve, where the x-axis will represent the number of clusters and the y-axis will be an evaluation metric. Let's say inertia for now.

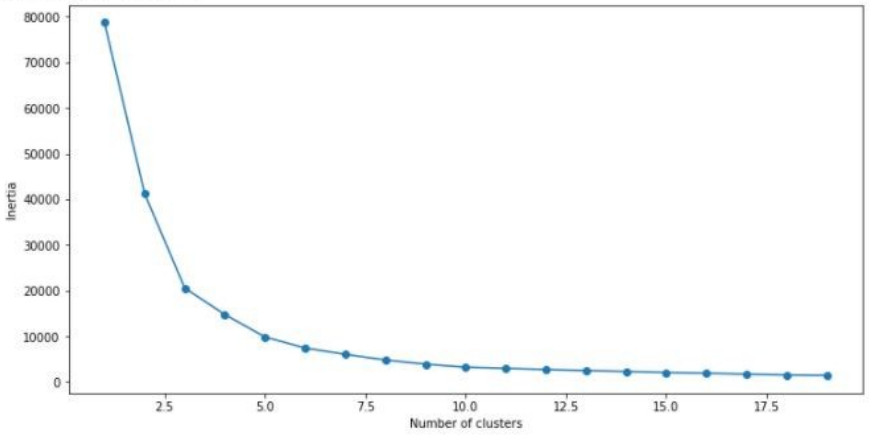
We got an inertia value of almost 2600. Now, let's see how we can use the elbow curve to determine the optimum number of clusters in Python.

We will first fit multiple k-means models and in each successive model, we will increase the number of clusters. We will store the inertia value of each model and then plot it to visualize the result:

```
[362] # fitting multiple k-means algorithms and storing the values in an empty list
      SSE = []
      for cluster in range(1,20):
          kmeans = KMeans(n_jobs = -1, n_clusters = cluster, init='k-means++')
          kmeans.fit(data_scaled)
          SSE.append(kmeans.inertia_)

      # converting the results into a dataframe and plotting them
      frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
      plt.figure(figsize=(12,6))
      plt.plot(frame['Cluster'], frame['SSE'], marker='o')
      plt.xlabel('Number of clusters')
      plt.ylabel('Inertia')
```

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



Color Array

+ Code + Text

Setting up plotter and color array

```
color_dict = dict({0:'brown',
                   1:'green',
                   2:'orange',
                   3:'red',
                   4:'dodgerblue',
                   5:'black',
                   6:'blue',
                   7:'pink',
                   8:'purple'})
```

Cluster Map, Clusters = 3

Initial clustering with two groups on the top and one group (green) on the bottom

The elbow is between 3 and 8.

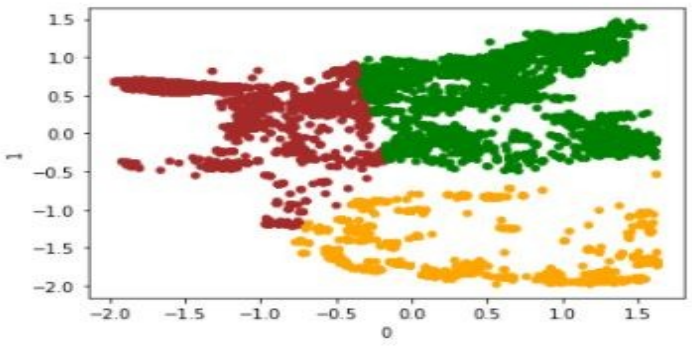
This is the cluster map for clusters = 3

```
# k means using 3 clusters and k-means++ initialization
kmeans = KMeans(n_jobs = -1, n_clusters = 3, init='k-means++')
kmeans.fit(data_scaled)
pred = kmeans.predict(data_scaled)

frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

```
1    17059
0    13789
2     8504
Name: cluster, dtype: int64
```

```
[365] ax1 = frame.plot.scatter(x=0, y=1, c=frame['cluster'].map(color_dict))
```



Cluster Map, Clusters = 4

Moving from 3 to 4 clusters, the two clusters on the top are now divided into 3 with a small part of the bottom cluster added to the center cluster.

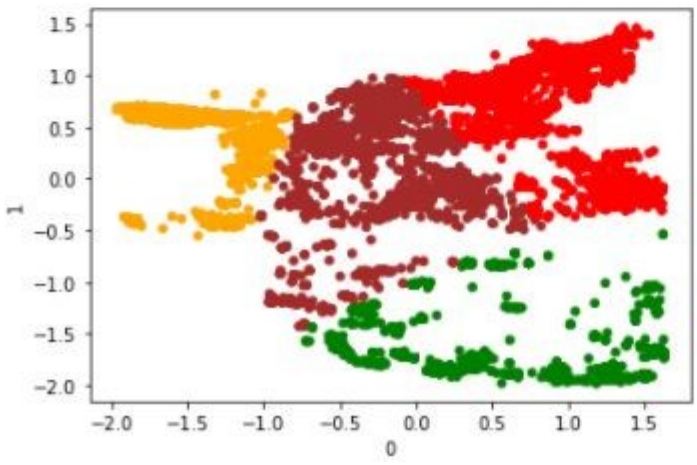
This is the cluster map for clusters = 4

```
[366] # k means using 4 clusters and k-means++ initialization
      kmeans = KMeans(n_jobs = -1, n_clusters = 4, init='k-means++')
      kmeans.fit(data_scaled)
      pred = kmeans.predict(data_scaled)

      frame = pd.DataFrame(data_scaled)
      frame['cluster'] = pred
      frame['cluster'].value_counts()
```

```
0    11560
3    11496
2     8482
1     7814
Name: cluster, dtype: int64
```

```
[367] ax1 = frame.plot.scatter(x=0, y=1, c=frame['cluster'].map(color_dict))
```



Cluster Map, Clusters = 5

Moving from 4 to 5 clusters, the bottom cluster is now divided in two, with the void in the middle of it as the main dividing area.

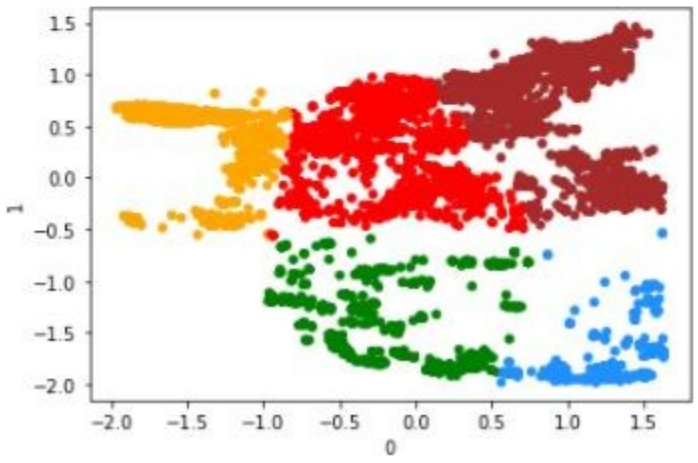
This is the cluster map for clusters = 5

```
[368] # k means using 5 clusters and k-means++ initialization
      kmeans = KMeans(n_jobs = -1, n_clusters = 5, init='k-means++')
      kmeans.fit(data_scaled)
      pred = kmeans.predict(data_scaled)

      frame = pd.DataFrame(data_scaled)
      frame['cluster'] = pred
      frame['cluster'].value_counts()
```

```
0    10946
3    10856
2     8532
4     4810
1     4208
Name: cluster, dtype: int64
```

```
[369] ax1 = frame.plot.scatter(x=0, y=1, c=frame['cluster'].map(color_dict))
```



Cluster Map, Clusters = 6

Moving from 5 to 6 clusters, the top right cluster is now divided in two, with the two 'fingers' of the cluster seperating

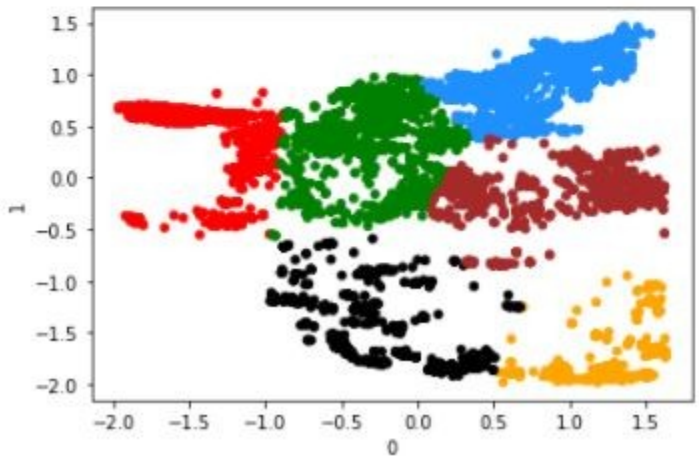
This is the cluster map for clusters = 6

```
[370] # k means using 6 clusters and k-means++ initialization
      kmeans = KMeans(n_jobs = -1, n_clusters = 6, init='k-means++')
      kmeans.fit(data_scaled)
      pred = kmeans.predict(data_scaled)

      frame = pd.DataFrame(data_scaled)
      frame['cluster'] = pred
      frame['cluster'].value_counts()
```

```
1    9033
4    9032
3    8499
2    4809
5    4009
0    3970
Name: cluster, dtype: int64
```

```
[371] ax1 = frame.plot.scatter(x=0, y=1, c=frame['cluster'].map(color_dict))
```



Cluster Map, Clusters = 7

Moving from 6 to 7 clusters, the upper center cluster is now divided in two, with the void in the middle of it as the main dividing area.

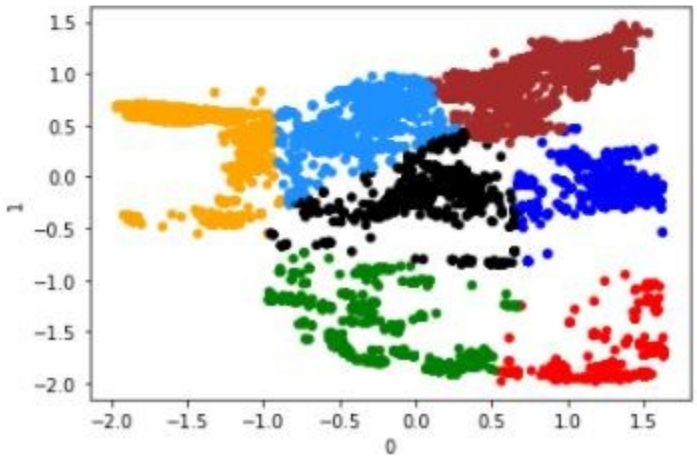
This is the cluster map for clusters = 7

```
# k means using 7 clusters and k-means++ initialization
kmeans = KMeans(n_jobs = -1, n_clusters = 7, init='k-means++')
kmeans.fit(data_scaled)
pred = kmeans.predict(data_scaled)

frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

```
0    8928
2    8498
4    6987
3    4808
5    4042
1    3943
6    2146
Name: cluster, dtype: int64
```

```
[373] ax1 = frame.plot.scatter(x=0, y=1, c=frame['cluster'].map(color_dict))
```



Cluster Map, Clusters = 8

Moving from 7 to 8 clusters, the top left cluster is now divided in two, with the two 'fingers' of the cluster separating.

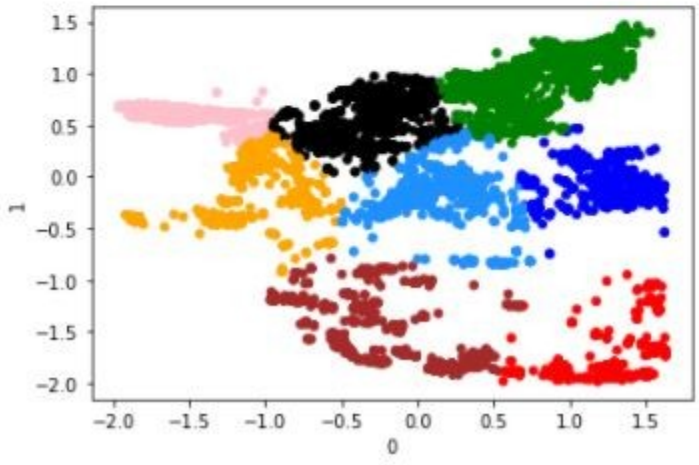
This is the cluster map for clusters = 8

```
[374] # k means using 8 clusters and k-means++ initialization
      kmeans = KMeans(n_jobs = -1, n_clusters = 8, init='k-means++')
      kmeans.fit(data_scaled)
      pred = kmeans.predict(data_scaled)

      frame = pd.DataFrame(data_scaled)
      frame['cluster'] = pred
      frame['cluster'].value_counts()
```

```
1    8877
5    6656
7    6388
3    4808
0    3928
4    3895
2    2745
6    2055
Name: cluster, dtype: int64
```

```
[375] ax1 = frame.plot.scatter(x=0, y=1, c=frame['cluster'].map(color_dict))
```



Video Link

Any in site about the data or the ICP in general

The ICP was quite enjoyable. Helped put context around the lecture from Tuesday.