Assignment 5
K Means Clustering

Date : 4th April,2019

Nanavati Tilak D.
(201811001)

K Means Clustering :

In the unsupervised learning when we do not have the labeled dataset at that time we use K Means clustering to determining the classes and the datasets belonging to that class.

Before applying the K Means algorithm we have to determine the value of K and we can do it with the help of 1) Elbow Method or 2).Datapoints Realization, here in this practical I am determining the value of K with datapoints realization method.

1. Code

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mat
import seaborn as sns

dataframe = pd.read_csv("kmeans.csv")

plt.scatter(dataframe.iloc[:,0],dataframe.iloc[:,1],marker=mat.markers.CARE
TDOWNBASE)
#sns.lmplot(x= "X1" , y= "X2" ,data=dataframe, fit_reg= False ,legend=
False,markers=["x","o","+"])
```

Output :

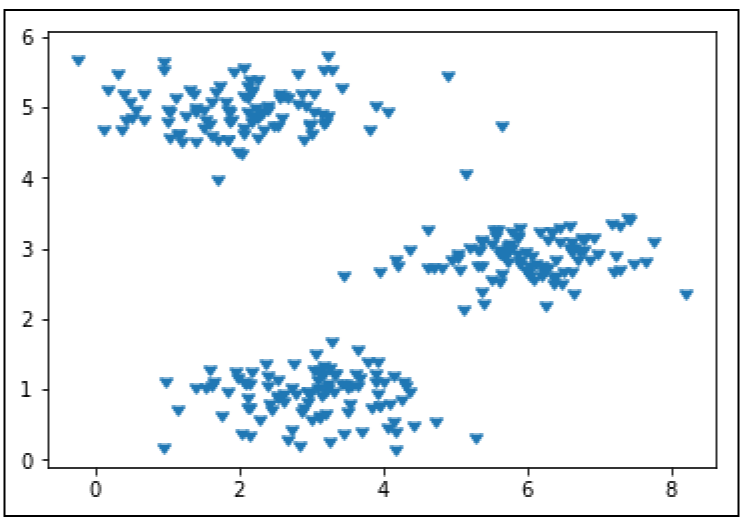


Fig. Datapoints having 3 Clusters

Observation :

From this we can observe that for the given dataset the value of K would be 3. And we can start our algorithm with K=3.

Computing Initial Means :

Code :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

data = pd.read_csv("kmeans.csv")
data
X = data[data.columns[0:2]]
#y = data[data.columns[4]]

X_train, X_test =
train_test_split(X,test_size=0.10,random_state=41)

"""a = []
b = []
c = []"""

#print(X_train)

"""for i in range(45):
    a.append(X_train.iloc[i])

for i in range(45,90):
    b.append(X_train.iloc[i])

for i in range(90,135):
    c.append(X_train.iloc[i])

"""
a = X_train.iloc[0:90,:]
```

```
b = X_train.iloc[91:180,:]
c = X_train.iloc[181:270,:]

ca = a.mean()
cb = b.mean()
cc = c.mean()

print(ca)
print(cb)
print(cc)
```

Output :

```
#Mean of Cluster A
X1    3.643222
X2    2.869687
#Mean of Cluster B
X1    3.624124
X2    3.239406
#Mean of Cluster C
X1    3.771684
X2    2.886196
```

Code :

```
import math as m

a = X_train.iloc[0:90,:]
b = X_train.iloc[91:180,:]
c = X_train.iloc[181:270,:]

for j in range(20):
    ca = a.mean()
    cb = b.mean()
    cc = c.mean()
```

```
    new_a = pd.DataFrame([[0, 0]], columns =
dataframe.columns[0:2])
    new_b = pd.DataFrame([[0, 0]], columns =
dataframe.columns[0:2])
    new_c = pd.DataFrame([[0, 0]], columns =
dataframe.columns[0:2])

    for i in range(270):
        p0 = X_train.iloc[i,0]
        p1 = X_train.iloc[i,1]

        dista = 0
        distb = 0
        distc = 0
        #print(ca.iloc[0])
        #Calculating the Euclidean Distance :
        dista = m.sqrt((p0-ca.iloc[0])**2+(p1-ca.iloc[1])**2)
        #print(dista)

        distb = m.sqrt((p0-cb.iloc[0])**2+(p1-cb.iloc[1])**2)
        #print(distb)

        distc = m.sqrt((p0-cc.iloc[0])**2+(p1-cc.iloc[1])**2)
        #print(distc)

        if(dista<distb and dista<distc):
            #print("**a")
            new_a = new_a.append(X_train.iloc[i,:])
        elif(distb<dista and distb<distc):
            #print("**b")
            new_b = new_b.append(X_train.iloc[i,:])
        elif(distc<distb and distc<dista):
            #print("**c")
            new_c = new_c.append(X_train.iloc[i,:])
```

```
    #print(new_a.append(X_train.iloc[1,:]))

    a = new_a
    b = new_b
    c = new_c

print("Total Number of Iterations required for Convergence :",j)
```

Output :

```
Total Number of Iterations required for Convergence : 19
```

Code :

```
plt.scatter(a.iloc[:,0],a.iloc[:,1])
plt.scatter(b.iloc[:,0],b.iloc[:,1])
plt.scatter(c.iloc[:,0],c.iloc[:,1])
```

Output :

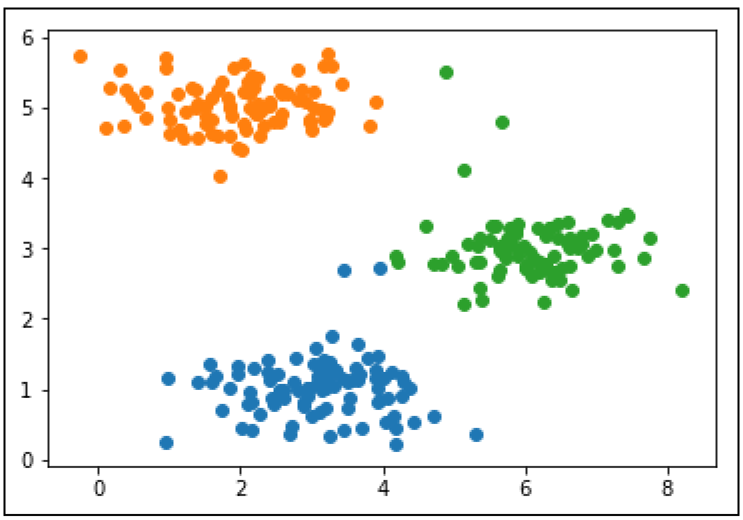


Fig. Identified Clusters

Now we repeat the experiment for DataSet 2 that was given in the logistic regression practical.

Code :

```
import pandas as pd
```

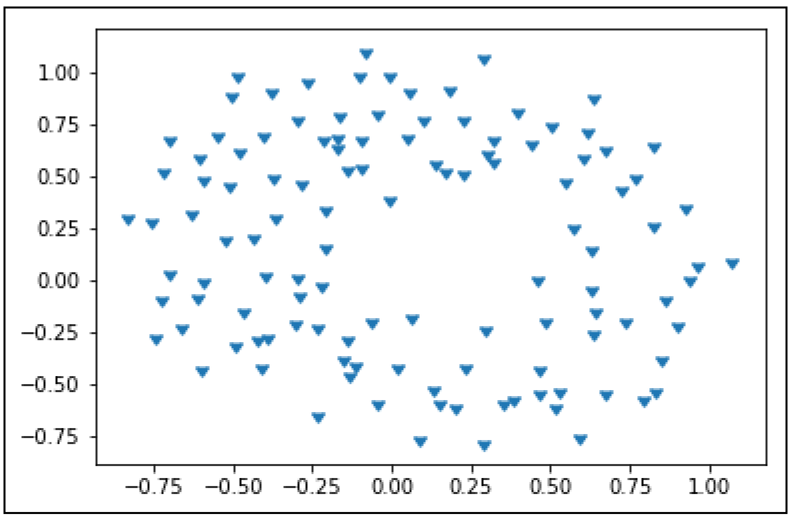
```
import matplotlib.pyplot as plt
import matplotlib as mat
import seaborn as sns

dataframe = pd.read_csv("ex2data2.csv")

plt.scatter(dataframe.iloc[:,0],dataframe.iloc[:,1],marker=mat.m
arkers.CARETDOWNBASE)

print(len(dataframe.iloc[:,0]))
```

Output :



Code :

```
import math as m



X = dataframe[dataframe.columns[0:2]]
X_train, X_test = train_test_split(X,test_size=0.10,random_state=41)

a = X_train.iloc[0:23,:]
b = X_train.iloc[24:46,:]
c = X_train.iloc[47:69,:]
```

```
d = X_train.iloc[70:93,:]
e = X_train.iloc[94:118,:]

for j in range(2):
    ca = a.mean()
    cb = b.mean()
    cc = c.mean()
    cd = d.mean()
    ce = e.mean()

    new_a = pd.DataFrame([[0, 0]], columns = dataframe.columns[0:2])
    new_b = pd.DataFrame([[0, 0]], columns = dataframe.columns[0:2])
    new_c = pd.DataFrame([[0, 0]], columns = dataframe.columns[0:2])
    new_d = pd.DataFrame([[0, 0]], columns = dataframe.columns[0:2])
    new_e = pd.DataFrame([[0, 0]], columns = dataframe.columns[0:2])

    #print("new_a",new_a)

    for i in range(106):
        p0 = X_train.iloc[i,0]
        p1 = X_train.iloc[i,1]

        dista = 0
        distb = 0
        distc = 0
        distd = 0
        diste = 0

        #print(ca.iloc[0])
        #Calculating the Euclidean Distance :
        dista = m.sqrt((p0-ca.iloc[0])**2+(p1-ca.iloc[1])**2)
        #print(dista)

        distb = m.sqrt((p0-cb.iloc[0])**2+(p1-cb.iloc[1])**2)
        #print(distb)

        distc = m.sqrt((p0-cc.iloc[0])**2+(p1-cc.iloc[1])**2)
        #print(distc)
```

```
        distd = m.sqrt((p0-cd.iloc[0])**2+(p1-cd.iloc[1])**2)
        #print(dista)

        diste = m.sqrt((p0-ce.iloc[0])**2+(p1-ce.iloc[1])**2)
        #print(dista)

        if(dista<distb and dista<distc and dista<distd and
dista<diste):
            #print("**a")
            #print("new_a_before")
            #print(new_a)
            new_a = new_a.append(X_train.iloc[i,0:2])
            #print("new_a_vector",new_a)
        elif(distb<dista and distb<distc and distb<distd and
distb<diste):
            #print("**b")
            new_b = new_b.append(X_train.iloc[i,0:2])
        elif(distc<distb and distc<dista and dista<distd and
dista<diste):
            #print("**c")
            new_c = new_c.append(X_train.iloc[i,0:2])
        elif(diste<dista and diste<distb  and diste<distc and
diste<distd):
            #print("**b")
            new_e = new_e.append(X_train.iloc[i,0:2])
        elif(distd<dista and distd<distb  and distd<distc and
distd<diste):
            #print("**c")
            new_d = new_d.append(X_train.iloc[i,0:2])
    #print(new_a.append(X_train.iloc[1,:]))

    a = new_a
    b = new_b
    c = new_c
    d = new_d
    e = new_e
```

```
print("Total Number of Iterations required for Convergence :",j)
print("Total Data Points in Class A :",len(a))
print("Total Data Points in Class B :",len(b))
print("Total Data Points in Class C :",len(c))
print("Total Data Points in Class D :",len(d))
print("Total Data Points in Class E :",len(e))
```

Output :

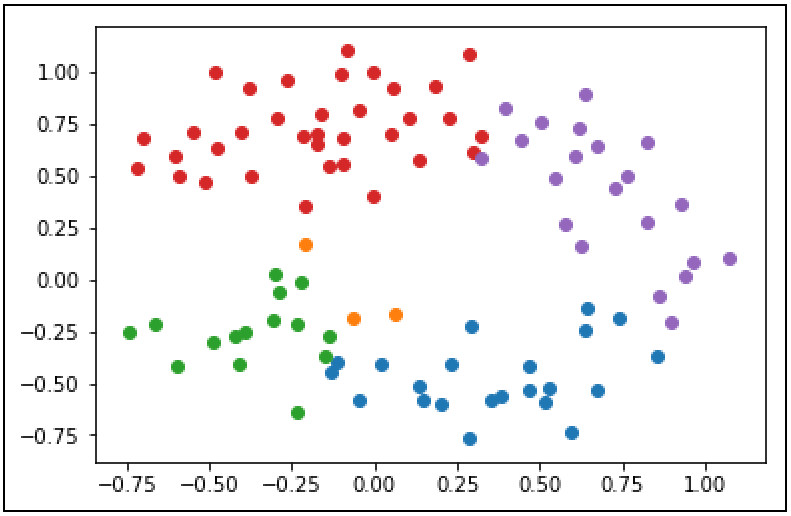


Fig. Identified Clusters

Code :

```
import math as m

all_dist = []
all_centroids = []

for epoch in range(100):

    X = dataframe[dataframe.columns[0:2]]
    X_train, X_test = 
train_test_split(X,test_size=0.10,random_state=(21+epoch+1)%11)

    a = X_train.iloc[0:23,:]
    b = X_train.iloc[24:46,:]
    c = X_train.iloc[47:69,:]
    d = X_train.iloc[70:93,:]
    e = X_train.iloc[94:118,:]

    for j in range(5):
        ca = a.mean()
        cb = b.mean()
        cc = c.mean()
        cd = d.mean()
        ce = e.mean()

        new_a = pd.DataFrame([[0, 0]], columns = 
dataframe.columns[0:2])
        new_b = pd.DataFrame([[0, 0]], columns = 
dataframe.columns[0:2])
        new_c = pd.DataFrame([[0, 0]], columns = 
dataframe.columns[0:2])
        new_d = pd.DataFrame([[0, 0]], columns = 
dataframe.columns[0:2])
        new_e = pd.DataFrame([[0, 0]], columns = 
dataframe.columns[0:2])
```

```
        #print("new_a",new_a)

        for i in range(106):
            p0 = X_train.iloc[i,0]
            p1 = X_train.iloc[i,1]

            dista = 0
            distb = 0
            distc = 0
            distd = 0
            diste = 0

            #print(ca.iloc[0])
            #Calculating the Euclidean Distance :
            dista =
m.sqrt((p0-ca.iloc[0])**2+(p1-ca.iloc[1])**2)
            #print(dista)

            distb =
m.sqrt((p0-cb.iloc[0])**2+(p1-cb.iloc[1])**2)
            #print(distb)

            distc =
m.sqrt((p0-cc.iloc[0])**2+(p1-cc.iloc[1])**2)
            #print(distc)

            distd =
m.sqrt((p0-cd.iloc[0])**2+(p1-cd.iloc[1])**2)
            #print(dista)

            diste =
m.sqrt((p0-ce.iloc[0])**2+(p1-ce.iloc[1])**2)
            #print(dista)
```

```
            if(dista<distb and dista<distc and dista<distd and 
dista<diste):
                #print("**a")
                #print("new_a_before")
                #print(new_a)
                new_a = new_a.append(X_train.iloc[i,0:2])
                #print("new_a_vector",new_a)
            elif(distb<dista and distb<distc and distb<distd and 
distb<diste):
                #print("**b")
                new_b = new_b.append(X_train.iloc[i,0:2])
            elif(distc<distb and distc<dista and dista<distd and 
dista<diste):
                #print("**c")
                new_c = new_c.append(X_train.iloc[i,0:2])
            elif(diste<dista and diste<distb  and diste<distc 
and diste<distd):
                #print("**b")
                new_e = new_e.append(X_train.iloc[i,0:2])
            elif(distd<dista and distd<distb  and distd<distc 
and distd<diste):
                #print("**c")
                new_d = new_d.append(X_train.iloc[i,0:2])
        #print(new_a.append(X_train.iloc[1,:]))

        a = new_a
        b = new_b
        c = new_c
        d = new_d
        e = new_e

    ca = a.mean()
    cb = b.mean()
    cc = c.mean()
    cd = d.mean()
```

```
    ce = e.mean()

    dista = 0
    distb = 0
    distc = 0
    distd = 0
    diste = 0

    #Calculating the Distances :
    for k in range(len(a)):
        dista =
m.sqrt((a.iloc[k,0]-ca.iloc[0])**2+(a.iloc[k,1]-ca.iloc[1])**2)
        #print(dista)
    for k in range(len(b)):
        distb =
m.sqrt((b.iloc[k,0]-cb.iloc[0])**2+(b.iloc[k,1]-cb.iloc[1])**2)
        #print(distb)
    for k in range(len(c)):
        distc =
m.sqrt((c.iloc[k,0]-cc.iloc[0])**2+(c.iloc[k,1]-cc.iloc[1])**2)
        #print(distc)
    for k in range(len(d)):
        distd =
m.sqrt((d.iloc[k,0]-cd.iloc[0])**2+(d.iloc[k,1]-cd.iloc[1])**2)
        #print(dista)
    for k in range(len(e)):
        diste =
m.sqrt((e.iloc[k,0]-ce.iloc[0])**2+(e.iloc[k,1]-ce.iloc[1])**2)
        #print(dista)

    all_dist.append(dista+distb+distc+distd+diste)

all_centroids.append([ca.iloc[0],ca.iloc[1],cb.iloc[0],cb.iloc[1
],cc.iloc[0],cc.iloc[1],cd.iloc[0],cd.iloc[1],ce.iloc[0],ce.iloc
[1]])
```

```
print("Total Number of Iterations required for Convergence :",j)
print("Total Data Points in Class A :",len(a))
print("Total Data Points in Class B :",len(b))
print("Total Data Points in Class C :",len(c))
print("Total Data Points in Class D :",len(d))
print("Total Data Points in Class E :",len(e))

print()
min_index = all_dist.index(min(all_dist))
print("Min Distance",all_dist[min_index])
print("Centroids",all_centroids[min_index])
```

Output :

```
Total Number of Iterations required for Convergence : 4
Total Data Points in Class A : 22
Total Data Points in Class B : 17
Total Data Points in Class C : 1
Total Data Points in Class D : 20
Total Data Points in Class E : 35

Min Distance : 0.7005372385027447
Centroids :
[0.6730841176470588, 0.38708135294117646, 0.0, 0.0, 0.13310277333333334,
0.7767046666666668, 0.0, 0.0, 0.46491153846153854, -0.4041245]
```

Code :

```
print(all_dist[all_dist.index(min(all_dist))])
```

Output :

```
0.5872662901366388
```