Project - Credit Card Segmentation

Introduction: This project is based on unsupervised learning technique of machine learning. In this, we need to divide the data in different and matching clusters to analyse the customers better. I used two Machine Learning Algorithms (k-means & mean shift) to analyse and predict the clusters for a dataset of "Customer Card Segmentation". I made this project on Jupyter Notebook. Also, I chose the value of k as 7 by using elbow method.

Technologies Used: Python, Machine Learning, k-means clustering algorithm, mean shift clustering algorithm, Matplotlib, seaborn, Joblib, etc.

Instructions: There are two files:

- CreditCardSegmentation.ipynb (main file)
- userPrediction.ipynb (for single input for users)

After saving the files in one single folder, just need to run the main file and all the output/prediction can be seen the main file only. Even the second file will also be running from the main file. To fit or train a model, our machines take a lot of time because of which I have saved my model with the help of 'joblib' so, it will not take much time to run now.

Code

File – CreditCardSegmentation.ipynb

Importing Libraries:

```
import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

%matplotlib inline

from matplotlib import pyplot as plt

import seaborn as sns
```

```
from sklearn import preprocessing
```

Reading csv file as Data Frame & performing some functions to know the data:

```
data = pd.read_csv("credit-card-data.csv")

data.info()

data.describe()

data.shape
```

Checking the data type of all the features and transform it if needed:

```
Data.dtypes()

#As above, everything is in int or float type except "CUST_ID" as it is in object type. But,
#we do not need to transform it as we do not need it in our model as well, so we can just
#drop that column.
```

Dropping columns which are not needed:

```
data = data.drop('CUST_ID', axis = 1)

data
```

Checking if there is any null value present in any of the feature in the dataset:

```
data.isnull().sum()

print("Count of every Features:\n", data.count())

print("\nPercentage of null values in every feature:\n", data.apply(lambda x:
sum(x.isnull()/len(data))*100))
```

As above, there are some missing values present in "CREDIT_LIMIT" & "MINIMUM_PAYMENT".
The percentage of all the null values is less than 5% so we can easily fill the null values.

In order to impute missing value, first we will have to check which method should we use:

```
df = data.copy()
```

```
#just checking any random value in the data of the feature and finalizing which method will be
#the best for imputing values:

actualValue = df['CREDIT_LIMIT'][3]

actualValue
```

1. Mean Imputing Method:

```
df['CREDIT_LIMIT'][3] = np.NAN

df['CREDIT_LIMIT'] = df['CREDIT_LIMIT'].fillna(df['CREDIT_LIMIT'].mean())

meanValue = df['CREDIT_LIMIT'][3]

meanValue
```

2. Median Imputing Method:

```
df['CREDIT_LIMIT'][3] = np.NAN

df['CREDIT_LIMIT'] = df['CREDIT_LIMIT'].fillna(df['CREDIT_LIMIT'].median())

medianValue = df['CREDIT_LIMIT'][3]

medianValue
```

As above, the closest imputed value is of MEAN method, so we can use mean imputing method to impute missing values in our dataset:

```
data['CREDIT_LIMIT'] = data['CREDIT_LIMIT'].fillna(data['CREDIT_LIMIT'].mean())

data['MINIMUM_PAYMENTS']=data['MINIMUM_PAYMENTS'].fillna(data['MINIMUM_
        PAYMENTS'].mean())

print("Count of every Features:\n", data.count(),"\n")

print("Null Values:\n",data.isnull().sum())
```

As above, all the missing values are imputed now. There is no null value in the dataset anymore.

VISUALIZATION

1. Visualizing every feature(distplot):

```
plt.figure(1, figsize = (15,35))

n = 0
```

```
for x in data.columns:
        n +=1
        plt.subplot(6, 3, n)
        plt.subplots_adjust(hspace = 0.5, wspace = 0.5)
        sns.distplot(data[x], bins = 20)
        plt.title('Distplot of {}'.format(x))
plt.show()
```

2. Count Plot for one of the feature(Tenure):

```
plt.figure(figsize = (15,5))
sns.countplot(y = 'TENURE', data = data)
plt.xticks(np.arange(0, 7500, step=300))
plt.show()
```

Feature Scaling

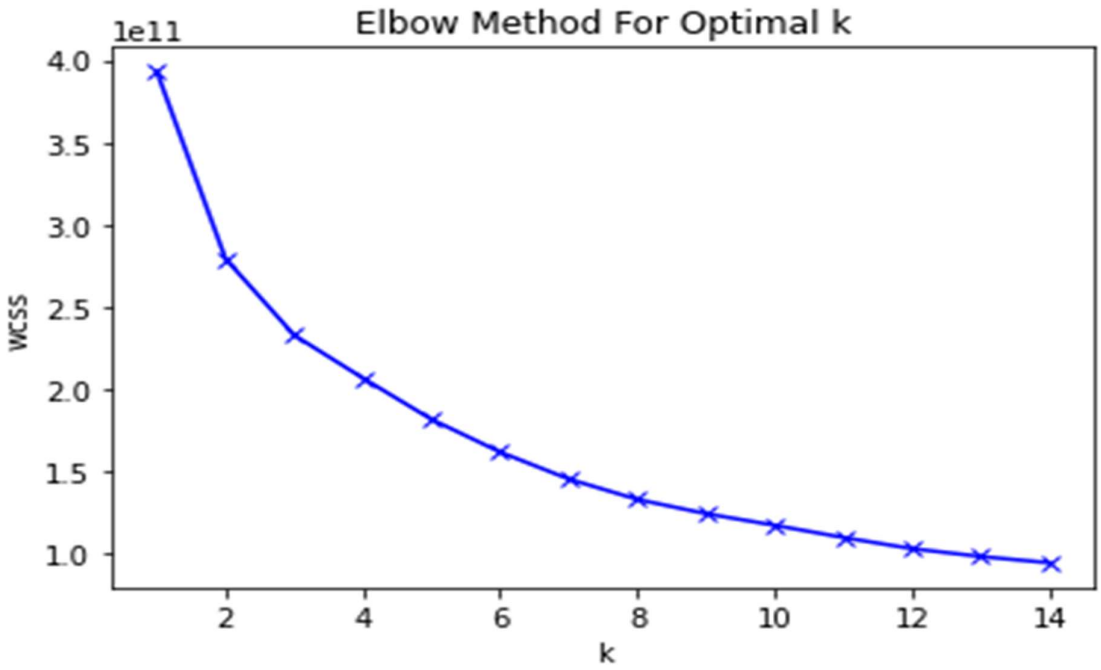
We need to normalise the data before aplying any algorithm because some features are in between 0-1 and some are very large like in thousands: This can cause false prediction as k-means algorithm works with distance. So, normalising all the features:

```
cols = data.values
min_max_scaler = preprocessing.MinMaxScaler()
cols_scaled = min_max_scaler.fit_transform(cols)
normalizeData = pd.DataFrame(cols_scaled, columns = data.columns)
normalizeData
```

Checking how many clusters we should use by ELBOW METHOD:

```
wcss = []
for k in range(1,15):
        km = KMeans(n_clusters=k)
          km = km.fit(data)
```

```
        wcss.append(km.inertia_)
plt.plot(range(1,15), wcss, 'bx-')
plt.xlabel('k')
plt.ylabel('wcss')
plt.title('Elbow Method For Optimal k')
plt.show()
```



As above, it looks like the number of optimal clusters are 7. So, k = 7

Training

```
msk = np.random.rand(len(data)) < 0.8
train = data[msk]
test = data[~msk]
```

1. k-means Clustering Algorithm

Calling kmeans fit method using clusters 7: and using the trained model, we will predict the clusters in the Test data:

```
def km():
        #SAVING THE MODEL BECAUSE WE DO NOT NEED TO TRAIN THE MODEL
        #AGAIN & AGAIN. SO, AFTER TRAINING SAVE THE MODEL.

        #TRAINING & PREDICTING

        kmeans = KMeans(n_clusters = 7, random_state = 0)

        kmeans.fit(train)

        joblib.dump(kmeans, 'km.pkl')

        km_model = joblib.load('km.pkl')

        km_predictions = km_model.predict(test)

        #Assigning prediction as a new column in the dataset:

        kmResult = test.copy()

        kmResult['PREDICTED_CLUSTER'] = km_predictions

        return kmResult
km_predictions = km()
```

Using groupby function to understand/analyze the model better:

```
output = km_predictions.groupby(by='PREDICTED_CLUSTER').mean()

output = output[normalizeData.columns]

output
```

Visualizing after prediction:

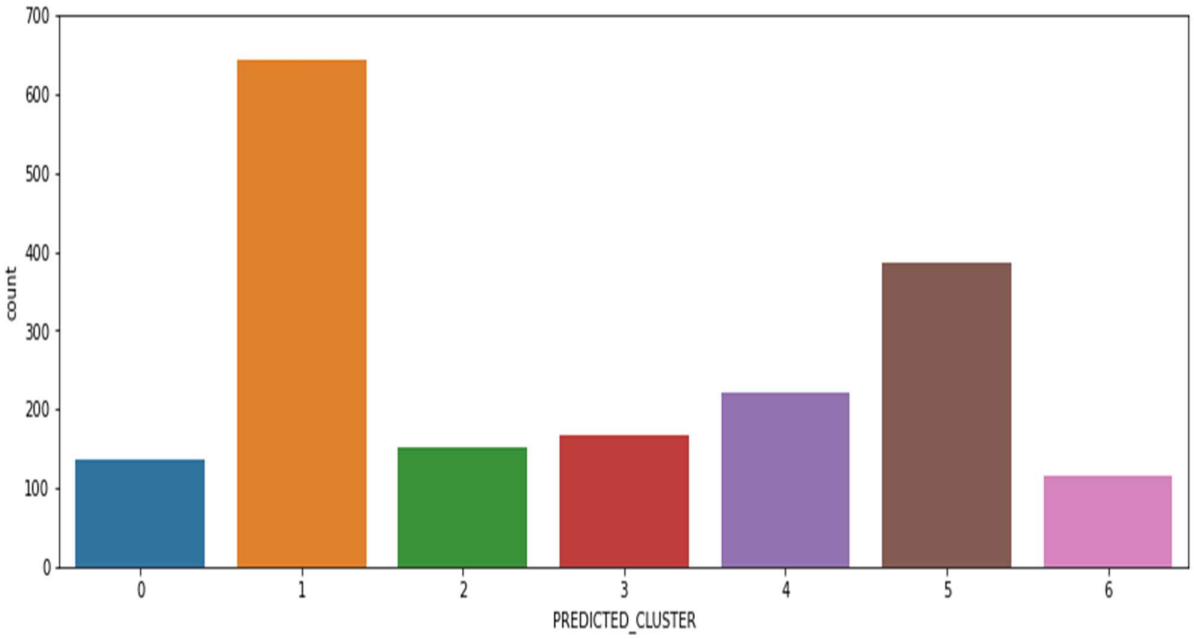
Analyzing Clusters: (which cluster carrying how many values from the test data)

```
plt.figure(figsize = (15,5))

sns.countplot(x = 'PREDICTED_CLUSTER', data = km_predictions)

plt.yticks(np.arange(0, 800, step = 100))

plt.show()
```



2. MeanShift Algorithm

```
def MS():

        #ms = MeanShift()

        #ms.fit(train)

        #joblib.dump(ms, 'ms.pkl')
```

```
    ms_model = joblib.load('ms.pkl')
    ms_predictions = ms_model.predict(test)
    msResult = test.copy()
    msResult['PREDICTED_CLUSTER'] = ms_predictions
    return msResult
ms_predictions = MS()
ms_predictions
```

Using groupby function to understand/analyze the model better:

```
output = km_predictions.groupby(by = 'PREDICTED_CLUSTER').mean()
output = output[normalizeData.columns]
output
```

Visualizing after prediction:

Analyzing Clusters: (which cluster carrying how many values from the test data)

```
plt.figure(figsize = (15,5))
sns.countplot(x = 'PREDICTED_CLUSTER', data = ms_predictions)
plt.yticks(np.arange(0, 1100, step = 100))
plt.show()
```

For user input:

```
%run userPrediction.ipynb
```

File – userPrediction.ipynb

Importing Libraries:

```
import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.cluster import MeanShift

import joblib
```

Reading data in DataFrame:

```
data = pd.read_csv("credit-card-data.csv")
```

Dropping columns which are not needed:

```
data = data.drop('CUST_ID', axis = 1)
```

Using Mean Method to fullfill the null values present in the dataset:

```
data['CREDIT_LIMIT'] = data['CREDIT_LIMIT'].fillna(data['CREDIT_LIMIT'].mean())

data['MINIMUM_PAYMENTS']=data['MINIMUM_PAYMENTS'].fillna(data['MINIMUM_
        PAYMENTS'].mean())
```

Taking input from User:

```
dictt = pd.DataFrame({'BALANCE' : None,

                      "BALANCE_FREQUENCY" : None,

                      "PURCHASES" : None,

                      "ONEOFF_PURCHASES" : None,

                      "INSTALLMENTS_PURCHASES" : None,

                      "CASH_ADVANCE" : None,
```

```
                "PURCHASES_FREQUENCY" : None,
                "ONEOFF_PURCHASES_FREQUENCY" : None,
                "PURCHASES_INSTALLMENTS_FREQUENCY" : None,
                "CASH_ADVANCE_FREQUENCY" : None,
                "CASH_ADVANCE_TRX" : None,
                "PURCHASES_TRX" : None,
                "CREDIT_LIMIT" : None,
                "PAYMENTS" : None,
                "MINIMUM_PAYMENTS" : None,
                "PRC_FULL_PAYMENT" : None,
                "TENURE": None
               }, index = [0])
def user():
        print("There are total of 17 features. Please fill in the values:")
        for i in range(0,len(data.columns)):
            inputValue = float(input("Enter value for "+"'"+data.columns[i]+"'"+": "))
            dictt[data.columns[i]] = inputValue
        return dictt
userInput = user()
```

Training

```
#np.random.seed(0)
msk = np.random.rand(len(normalizeData)) < 0.8
train = data[msk]
test = data[~msk]
```

```
Appending userInput in the test data:

test = test.append(userInput, ignore_index = True)
```

k-means Clustering Algorithm:

Calling kmeans fit method using clusters 7: and using the trained model, we will predict the clusters in the Test data:

```
def km():
        #SAVING THE MODEL BECAUSE WE DO NOT NEED TO TRAIN THE MODEL
            #AGAIN & AGAIN. SO, AFTER TRAINING SAVE THE MODEL.
          #TRAINING & PREDICTING
          kmeans = KMeans(n_clusters = 7, random_state = 0)
          kmeans.fit(train)
          joblib.dump(kmeans, 'km_userInput.pkl')
          km_model = joblib.load('km_userInput.pkl')
          km_predictions = km_model.predict(test)
          #Assigning prediction as a new column in the dataset:
          kmResult = test.copy()
          kmResult['PREDICTED_CLUSTER'] = km_predictions
          return kmResult
km_predictions = km()


prediction = km_predictions['PREDICTED_CLUSTER'][len(km_predictions)-1]
print("The values entered by you are lying in Cluster:",prediction)
```