Machine Learning

Assignment-4

Laxma Reddy Nalla

700732071

Github Link: CS-5710/Assignment-4 at dev · LaxmaReddy-Nalla/CS-5710 (github.com)

Youtube Link: https://youtu.be/w7Ibdekp0so

Question1:

1. Apply Linear Regression to the provided dataset using underlying steps.

    a. Import the given “Salary_Data.csv”

    b. Split the data in train_test partitions, such that 1/3 of the data is reserved as test subset.

    c. Train and predict the model.

    d. Calculate the mean_squared error

    e. Visualize both train and test data using scatter plot.

For this problem I’m using Salary Dataset on this Dataset I will be applying LinearRegression model to predict the Salary based on the persons experience.

```
# Importing Dataset and printing sample of the dataset
salary_df = pd.read_csv('Salary_Data.csv')
salary_df.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

```
# Splittingo the dataset into Independent and dependent Variables
x = salary_df.iloc[:,:-1].values
y = salary_df.iloc[:,-1]
```

Importing Dataset and printing sample of data. Splitting Dataset into independent and dependent variables.

```
# Splitting the dataset into train and test split datasets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=1/3, random_state=0)
```

```
# Importing the LinearRegression Model from sklearn.linear_model
# Training and predicting vallues on the LinearRegression model
from sklearn.linear_model import LinearRegression
classifier = LinearRegression()
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
```

```
# Calculating the mean_squred_error the model predicted values and actual values
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
mse
```

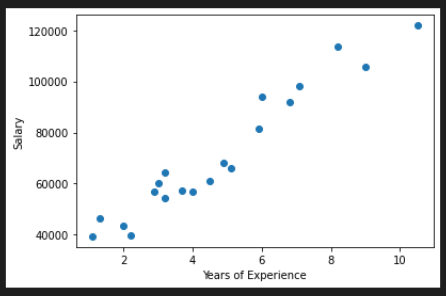
```
21026037.329511296
```

Splitting Data Into training and testing datasets. Importing LinearRegression model from sklearn.model_selection then training and model evaluation.

Evaluting mean_square_error for actual value and predicted value

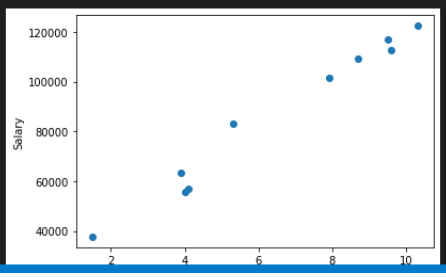
```
# Plotting the scatter graph for training and test datasets
import matplotlib.pyplot as plt
%matplotlib inline
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.scatter(x_train, y_train)
```

```
<matplotlib.collections.PathCollection at 0x7f2b85c78610>
```




```
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.scatter(x_test, y_test)
```

```
<matplotlib.collections.PathCollection at 0x7f2b85d1b9d0>
```



Plotting Train and test Dataset using scatter plot

Question 2:

2. Apply K means clustering in the dataset provided:

• Remove any null values by the mean.

• Use the elbow method to find a good number of clusters with the K-Means algorithm

• Calculate the silhouette score for the above clustering

```
# importing basic essential modules
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

```
# importing dataset printing sample data
k_df = pd.read_csv('K-Mean_Dataset.csv')
k_df.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	0.000000
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	0.000000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	1.000000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	0.083333
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	0.083333

Importing Dataset using pandas

```
# Getting Statical Overview of Dataset
k_df.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FR
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	89
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	

```
# Checking Sum of Null Values in DataSet
k_df.isnull().sum()
```

```
CUST_ID                           0
BALANCE                           0
BALANCE_FREQUENCY                 0
PURCHASES                         0
ONEOFF_PURCHASES                  0
INSTALLMENTS_PURCHASES            0
CASH_ADVANCE                      0
PURCHASES_FREQUENCY               0
ONEOFF_PURCHASES_FREQUENCY        0
PURCHASES_INSTALLMENTS_FREQUENCY  0
CASH_ADVANCE_FREQUENCY            0
```

Getting Statical information using Describe Function

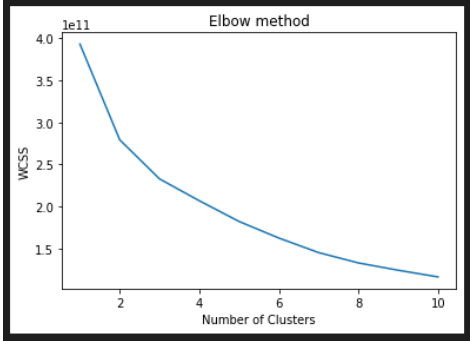
Checking no of null values in each column.

```
# Removing the First column Which is not required for model training and prediction
# FIlling null values with mean values of that column
k_df = k_df.iloc[:,1:]
k_df['MINIMUM_PAYMENTS'] = k_df['MINIMUM_PAYMENTS'].fillna(np.mean(k_df['MINIMUM_PAYMENTS']))
k_df['CREDIT_LIMIT'] = k_df['CREDIT_LIMIT'].fillna(np.mean(k_df['CREDIT_LIMIT']))
k_df.isnull().sum()
```

```
BALANCE_FREQUENCY                 0
PURCHASES                         0
ONEOFF_PURCHASES                  0
INSTALLMENTS_PURCHASES            0
CASH_ADVANCE                      0
PURCHASES_FREQUENCY               0
ONEOFF_PURCHASES_FREQUENCY        0
PURCHASES_INSTALLMENTS_FREQUENCY  0
CASH_ADVANCE_FREQUENCY            0
CASH_ADVANCE_TRX                  0
PURCHASES_TRX                     0
CREDIT_LIMIT                      0
PAYMENTS                          0
MINIMUM_PAYMENTS                  0
PRC_FULL_PAYMENT                  0
TENURE                            0
dtype: int64
```

FIlling null values using the mean of the column

```
# Generating Graph for Number of clusters using Elbow method.

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
WCSS = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
    kmeans.fit(k_df)
    WCSS.append(kmeans.inertia_)

plt.plot(range(1,11),WCSS)
plt.title('Elbow method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



Checking Elbow graph to determine no of clusters here I recognise 2 clusters.

```
# From elbow method Taking cluster count as 3
# training the KMeans Clustering Algorithm using no of clusters which I have analysed using elbow method
clusters = 3
kmeans = KMeans(n_clusters=clusters)
kmeans.fit(k_df)
y_kmeans = kmeans.predict(k_df)

# Calculating Silhouette score for the KMeans model
from sklearn.metrics import silhouette_score
score = silhouette_score(k_df, y_kmeans)
print("Silhouette Score: ", score)
```

```
Silhouette Score:  0.4636666618132307
```

Here I calculated Silhouette Score: 0.4636666618132307

QUestion 3:

3. Try feature scaling and then apply K-Means on the scaled features. Did that improve the Silhouette score? If Yes, can you justify why

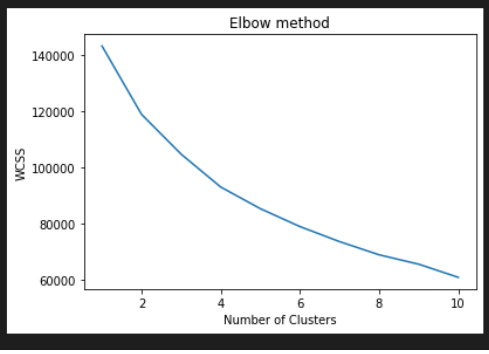
- using Feature scaling for the K-Mean Dataset and after feature scaling Apply Kmeans algorithm and calculate Silhouette score\

```
# importing Standard Scaler from sklearn preprocessing to feature scale the Dateset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# scaling the dataset
scaled_k_df = pd.DataFrame(scaler.fit_transform(k_df), columns=k_df.columns)
scaled_k_df.head()
```

	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_
0	-0.249434	-0.424900	-0.356934	-0.349079	-0.466786	-0.806490	-0.678661	-0.707313	
1	0.134325	-0.469552	-0.356934	-0.454576	2.605605	-1.221758	-0.678661	-0.916995	
2	0.518084	-0.107668	0.108889	-0.454576	-0.466786	1.269843	2.673451	-0.916995	
3	-1.016953	0.232058	0.546189	-0.454576	-0.368653	-1.014125	-0.399319	-0.916995	
4	0.518084	-0.462063	-0.347294	-0.454576	-0.466786	-1.014125	-0.399319	-0.916995	

Scaling the dataset using Standardscaler

```
# Evaluting the no of clusters for the scaled dataset using Elbow Method
scaled_WCSS = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
    kmeans.fit(scaled_k_df)
    scaled_WCSS.append(kmeans.inertia_)

# Plotting Graph for elbow method
plt.plot(range(1,11),scaled_WCSS)
plt.title('Elbow method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



Plotting Elbow graph to determine the no of clusters

```
    # From elbow method considering clusters as 4
    scaled_clusters = 4
    kmeans = KMeans(n_clusters=clusters)
    kmeans.fit(scaled_k_df)
    scaled_y_kmeans = kmeans.predict(scaled_k_df)
    scaled_y_kmeans
    s_score = silhouette_score(scaled_k_df, y_kmeans)
    print("Silhouette Score for Scaled DataSet: ", s_score)
```

```
Silhouette Score:  0.16973172721852334
```

Silhouette Score : 0.16973172721852334

Conclusion:

Here I observed that after scaling the dataset using standard scaler, I got the lesser Silhouette score. The silhouette score is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to

1: Means clusters are well apart from each other and clearly distinguished.

0: Means clusters are indifferent, or we can say that the distance between clusters is not significant.

I observed that by scaling the features the differential value between each datapoint will be reduced by that the model will not be able to make clusters accurately and the inter-cluster distance will be reduced and the model struggle to generalize.