MACHINE LEARNING ASSIGNMENT 5

NAME - VISHNU PONUGOTI
700744508

Video link -
https://drive.google.com/file/d/1z2oC4zIGZawrqdAntiOMAxse_NNJPjtI/view?usp=share_link

1. Principal Component Analysis

a. Apply PCA on CC dataset.

To do data analysis and apply machine learning algorithms on data, first I imported a few python libraries.

Using the read_csv method imported the “CC“ data set. The head() method of pandas library results top most rows of a data set

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
#1 # Reading cc data set
df= pd.read_csv("C:\\Users\\vishn\\OneDrive\\Desktop\\CC GENERAL.csv")

# Results top most rows in a data set
df.head()
```

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

Isnull() method of pandas library checks for any values present in the dataset.

To perform PCA on this data set we don’t need the output labels because PCA does not rely on the output labels. Using the drop method, we removed a few columns which are unnecessary

```
# Checking any null values are present
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
CASH_ADVANCE_TRX                     0
PURCHASES_TRX                        0
CREDIT_LIMIT                         1
PAYMENTS                             0
MINIMUM_PAYMENTS                   313
PRC_FULL_PAYMENT                     0
TENURE                               0
dtype: int64
```

```
mean1=df['CREDIT_LIMIT'].mean()
mean2=df['MINIMUM_PAYMENTS'].mean()

 # replacing null values with mean of a column
df['CREDIT_LIMIT'].fillna(value=mean1, inplace=True)

df['MINIMUM_PAYMENTS'].fillna(value=mean2, inplace=True)
```

From sklearn python library we imported the PCA method to perform PCA on the data set. PCA results in a data frame with features having maximum variance with other features by ignoring the duplicate features. Here we reduced the dimensionality of data into two components by keeping k value is equal to 2.

```
In [8]: # Preprocessing the data by removing the columns
        X = df.drop(['TENURE','CUST_ID'],axis=1).values
        y = df['TENURE'].values
```

```
In [9]: # Performing PCA
        pca2 = PCA(n_components=2)

        # pca is applied on the data set without output labels
        principalComponents = pca2.fit_transform(X)

        # Creating a data frame for the pca results
        principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])

        # Adding a new column to the data frame
        finalDf = pd.concat([principalDf, df[['TENURE']]], axis = 1)

        # Printing the results
        finalDf
```

Out[9]:

	principal component 1	principal component 2	TENURE
0	-4326.383979	921.566882	12
1	4118.916665	-2432.846346	12
2	1497.907641	-1997.578694	12
3	1394.548536	-1488.743453	12
4	-3743.351896	757.342657	12
...	...	...	...
8945	-4208.357725	1122.443291	6
8946	-4123.923788	951.683820	6
8947	-4379.443989	911.504583	6
8948	-4791.117531	1032.540961	6
8949	-3623.702535	1555.134786	6

8950 rows × 3 columns

b. Apply k-means algorithm on the PCA result and report your observation if the silhouette score has improved or not?

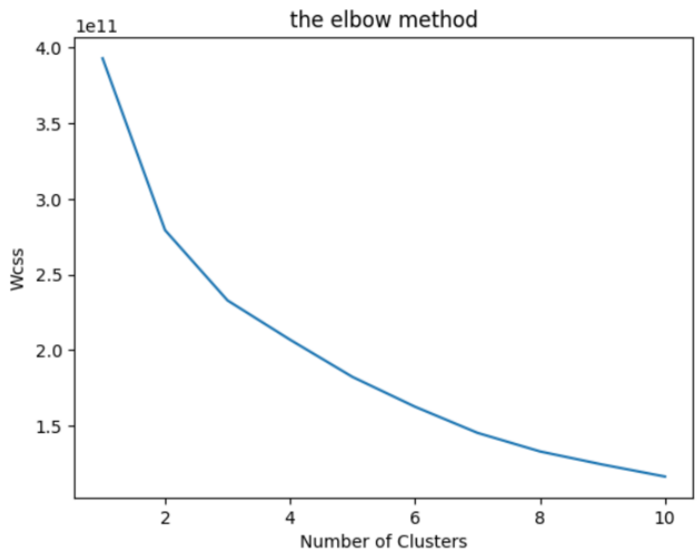
To perform k-means algorithm on a data set first we need to find the number of clusters required to fit our data together into clusters by using elbow method. This elbow method results in a graph from where we need to find the number of clusters value i.e., k value.

From the graph below, from number of clusters is 2 the wcss value starts decreasing linearly. So, the number of clusters required to fit our data is 3 i.e., k value is 3. In k-means algorithm k is the number of clusters.

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

from sklearn.cluster import KMeans
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(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.title('the elbow method')
plt.xlabel('Number of Clusters')
plt.ylabel('Wcss')
plt.show()
```



Using KMeans method of sklearn library, I applied K- Means algorithm on data set we got after performing PCA. After performing k-means on the PCA data we got a silhouette score of 57% which is higher than the silhouette score of raw data without performing PCA.

The silhouette score has been improved when we perform PCA on the data set. when we applied kmeans on the data set without performing PCA we got a silhouette score of 46.5%. After performing PCA we got a silhouette score of 57%. The silhouette score has been improved by more than 10%.

```
#  Calculate the silhouette score for the above clustering
# this is the k in kmeans
nclusters = 3
km = KMeans(n_clusters=nclusters)

# fitting out kmeans model with our data set
km.fit(finalDf)

y_cluster_kmeans = km.predict(finalDf)
from sklearn import metrics
score = metrics.silhouette_score(finalDf, y_cluster_kmeans)
print(score)
```

```
0.5720391530020281
```

C. Perform Scaling + PCA + K-Means and report performance.

Using StandardScalar method we performed feature scaling on the data set. Feature scaling is used to normalize the range of all features.

We are performing PCA on the feature scaled data set using the PCA method.

```
# Feature scaling using standard scaler
scaler = StandardScaler()
X_Scale = scaler.fit_transform(X)
```

```
# Performing pca
pca3 = PCA(n_components=2)
principalComponents1 = pca3.fit_transform(X_Scale)

principalDf1 = pd.DataFrame(data = principalComponents1, columns = ['principal component 1', 'principal component 2'])

finalDf2 = pd.concat([principalDf1, df[['TENURE']]], axis = 1)
finalDf2
```

3]:

	principal component 1	principal component 2	TENURE
0	-1.718893	-1.072937	12
1	-1.169306	2.509310	12
2	0.938413	-0.382590	12
3	-0.907503	0.045857	12
4	-1.637831	-0.684969	12
...	...	...	...
8945	-0.025277	-2.034124	6
8946	-0.233113	-1.656651	6
8947	-0.593880	-1.828113	6
8948	-2.007671	-0.673771	6
8949	-0.217931	-0.418502	6

8950 rows × 3 columns

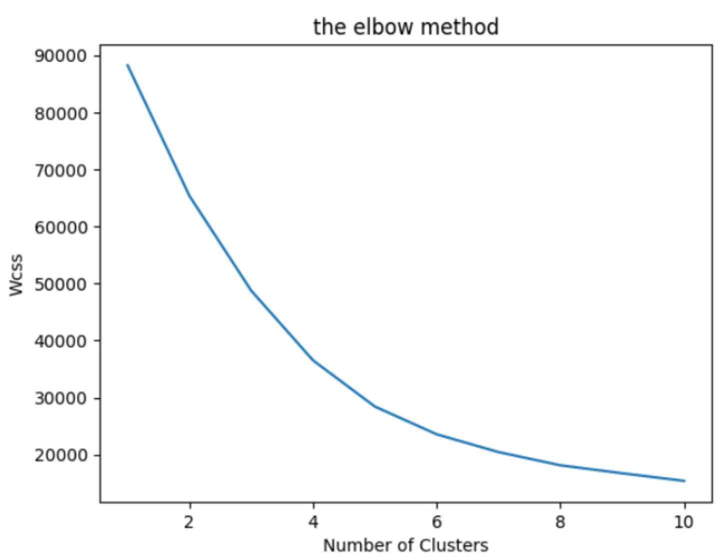
To perform k-means algorithm on a data set first we need to find the number of clusters required to fit our data together into clusters by using elbow method.

The elbow method results in a graph. From graph, the next point to point where the wcss value starts decreasing linearly will be the k value. From the graph below, from number of clusters is 2 the wcss value starts decreasing linearly. So, the number of clusters required to fit our data is 3 i.e., k value is 3.

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

from sklearn.cluster import KMeans
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(finalDf2)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.title('the elbow method')
plt.xlabel('Number of Clusters')
plt.ylabel('Wcss')
plt.show()
```



Using KMeans method of sklearn library, I applied K- Means algorithm by taking k value as 3 on data set, we got after performing feature scaling and

PCA. After performing k-means on this data we got a silhouette score of 38%.

```
#  Calculate the silhouette score for the above clustering
# this is the k in kmeans
nclusters = 3
km = KMeans(n_clusters=nclusters)
km.fit(finalDf2)

y_cluster_kmeans = km.predict(finalDf2)
from sklearn import metrics
score = metrics.silhouette_score(finalDf2, y_cluster_kmeans)
print(score)
```

```
0.3824521107736649
```

2. Use pd_speech_features.

csv Using read_csv method imported a csv file. The head() method of pandas library results top most rows of a data set.

```
# Reading pd_speech_features csv file
df1= pd.read_csv("C:\\Users\\vishn\\OneDrive\\Desktop\\pd_speech_features.csv")
df1.head()
```

]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kurtosisValue_dec_2
0	0	1	0.85247	0.71826	0.57227	240	239	0.008064	0.000087	0.00218	...	1.562
1	0	1	0.76686	0.69481	0.53966	234	233	0.008258	0.000073	0.00195	...	1.558
2	0	1	0.85083	0.67604	0.58982	232	231	0.008340	0.000060	0.00176	...	1.564
3	1	0	0.41121	0.79672	0.59257	178	177	0.010858	0.000183	0.00419	...	3.780
4	1	0	0.32790	0.79782	0.53028	236	235	0.008162	0.002669	0.00535	...	6.172

5 rows × 755 columns

a. Perform Scaling

Using StandardScalar method we performed feature scaling on the data set. Feature scaling is used to normalize the range of all features.

b. Apply PCA (k=3)

To perform PCA on this data set we don't need the output labels because PCA does not rely on the output labels. Using the drop method, we removed a class column which is unnecessary.

```
In [17]:  # Preprocessing the data
          X = df1.drop('class',axis=1).values
          y = df1['class'].values
```

```
In [18]:  # Performing feature selection
          scaler = StandardScaler()
          X_Scale = scaler.fit_transform(X)
```

```
In [19]:  # Performing pca
          pca4 = PCA(n_components=3)
          principalComponents2 = pca4.fit_transform(X_Scale)

          principalDf2 = pd.DataFrame(data = principalComponents2, columns = ['principal component 1', 'principal component 2',
                                                                              'principal components 3'])
          finalDf3 = pd.concat([principalDf2, df1[['class']]], axis = 1)
          finalDf3
```

Out[19]:

	principal component 1	principal component 2	principal components 3	class
0	-10.047372	1.471077	-6.846404	1
1	-10.637725	1.583748	-6.830978	1
2	-13.516185	-1.253542	-6.818698	1
3	-9.155083	8.833600	15.290902	1
4	-6.764470	4.611467	15.637121	1
...	...	...	...	...
751	22.322682	6.481911	1.458753	0
752	13.442877	1.449412	9.352295	0
753	8.270264	2.391285	-0.908674	0
754	4.011760	5.412255	-0.847133	0
755	3.993114	6.072416	-2.020725	0

756 rows × 4 columns

C. Use SVM to report performance

sklearn module contains train_test_split method to split our data set into training and testing data sets. In this method, test_size defines how much proportion of data to be in the test data set. When we change test_size value whole analysis results will change.

Support vector machine algorithm is applied to the data set we got after performing PCA using sklearn module. We got an accuracy of 74.8% when we trained SVM on our data set.

```
# Splitting our data into training and testing part
X_train, X_test, y_train, y_true = train_test_split(finalDf3[::-1], finalDf3['class'], test_size = 0.30, random_state = 0)
```

```
# Training and predcting svm model on our data set
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Support Vector Machine's
from sklearn.svm import SVC

classifier = SVC()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_true, y_pred))
print(confusion_matrix(y_true, y_pred))

# Accuracy score
from sklearn.metrics import accuracy_score
print('accuracy is',accuracy_score(y_pred,y_true))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        57
           1       0.75      1.00      0.86       170

    accuracy                           0.75       227
   macro avg       0.37      0.50      0.43       227
weighted avg       0.56      0.75      0.64       227

[[  0  57]
 [  0 170]]
accuracy is 0.748898678414097
```

3. Apply Linear Discriminant Analysis (LDA) on Iris.csv dataset to reduce dimensionality of data to k=2.

A csv file was imported using the read_csv method. The top rows of a data set are returned by the pandas library's head() method.

```
# Reading iris csv file
df2= pd.read_csv("C:\\Users\\vishn\\OneDrive\\Desktop\\Iris.csv")

df2.head()
```

!3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Isnull() method of pandas library checks for any values present in data set. In this iris dataset there are no null values.

```
# Checking null values
df2.isnull().any()
```

```
4]: Id               False
    SepalLengthCm    False
    SepalWidthCm     False
    PetalLengthCm    False
    PetalWidthCm     False
    Species          False
    dtype: bool
```

To perform LDA on this data set we need the output labels because LDA rely on these output labels to reduce the dimensionality of data based on output classes.

```
# Preprocessing the data
X = df2.iloc[:, 1:5].values
y = df2.iloc[:, 5].values
```

The LinearDiscriminantAnalysis class of the sklearn.discriminant_analysis library can be used to Perform LDA in Python. By setting n_components value as 2 we will get the results in two linear discriminates. We execute the fit and transform methods to retrieve our results.

```
# performing lda on the data set
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=2)
LinearDA = lda.fit_transform(X, y)

# Converting our results into a dataset
LinearDf = pd.DataFrame(data = LinearDA, columns = ['LD 1', 'LD 2'])

# Appending species column to the data frame
finalLda = pd.concat([LinearDf, df2[['Species']]], axis = 1)
finalLda
```

6]:

	LD 1	LD 2	Species
0	8.084953	0.328454	Iris-setosa
1	7.147163	-0.755473	Iris-setosa
2	7.511378	-0.238078	Iris-setosa
3	6.837676	-0.642885	Iris-setosa
4	8.157814	0.540639	Iris-setosa
...	...	...	...
145	-5.674013	1.661346	Iris-virginica
146	-5.197129	-0.365506	Iris-virginica
147	-4.981712	0.812973	Iris-virginica
148	-5.901486	2.320751	Iris-virginica
149	-4.684009	0.325081	Iris-virginica

150 rows × 3 columns

4. Briefly identify the difference between PCA and LDA

Dimensionality reduction in machine learning refers to the process of collecting a collection of major variables to reduce the number of random variables being considered. Principle Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are two main algorithms in dimensionality reduction.

PCA is unsupervised while LDA is a supervised dimensionality reduction technique.

PCA gets the results without depending on the output labels. PCA results a data set with maximum variance between the features by ignoring the duplicates of other features. Since the variance between the features is independent of the outcome, PCA does not consider the output labels.

LDA depends on the output labels. Based on the output labels information LDA reduces the feature set dimensions and finds a decision boundary. The data points are then projected to new dimensions so that the clusters are as distinct from one another as possible, and the individual components of a cluster are as near the cluster centroid as possible