Assignment 5

Name: Rohitha Sai Alla

No: 700734780

Video link: https://drive.google.com/file/d/1GBBaIJAopDgIa5oZ9TqA-Q5D2wK8iTcd/view?usp=sharing

Git link: https://github.com/RohithaSaiML/Assignment5

Principal Component Analysis

In [35]:
```
#dropping cust_id as it is just primary key and may effect the process
dataset = dataset.drop("CUST_ID", axis=1)
```

Out[35]:
```
CUST_ID                             False
BALANCE                             False
BALANCE_FREQUENCY                   False
PURCHASES                           False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
CASH_ADVANCE                        False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
CREDIT_LIMIT                         True
PAYMENTS                            False
MINIMUM_PAYMENTS                     True
PRC_FULL_PAYMENT                    False
TENURE                              False
dtype: bool
```

CREDIT_LIMIT and MINIMUM_PAYMENTS have null values. And these can be made(assume) zero because minimum a person can pay is zero and if minimum payment is done then credit limit also equals to zero.

1.a Apply PCA on CC dataset

```
In [72]: #1.a applying pca on cc dataset
pca = PCA(3)
x_pca = pca.fit_transform(x)
principalDf = pd.DataFrame(data = x_pca, columns = ['principal comp. 1', 'principal comp. 2', 'principal comp. 3'])
finalDf = pd.concat([principalDf, dataset.iloc[:,-1]], axis = 1)
finalDf.head()
```

Output:

Out[72]:

	principal comp. 1	principal comp. 2	principal comp. 3	TENURE
0	-4020.582516	1021.360799	-114.092023	12
1	3656.477239	-1886.785629	3542.535287	12
2	1257.505760	-2435.136919	-1677.837683	12
3	1307.461232	-2160.027698	-2503.293406	12
4	-3647.914965	1075.020369	54.820035	12

1.b Applying kmeans on PCA result

```
In [80]: #1.b applying kmeans on pca result
X = finalDf.iloc[:,0:-1]
y = finalDf.iloc[:,-1]

nclusters = 3 #k value
km = KMeans(n_clusters=nclusters)
km.fit(X)

# predicting the cluster for each value
y_cluster_kmeans = km.predict(X)


# prediction summary of kmeans
print(classification_report(y, y_cluster_kmeans, zero_division=1))
print(confusion_matrix(y, y_cluster_kmeans))


train_accuracy = accuracy_score(y, y_cluster_kmeans)
print("\nAccuracy for our Training dataset with PCA:", train_accuracy)
```

Calculating silhouette score

```
#calculating silhouette score
score = metrics.silhouette_score(X, y_cluster_kmeans)
print("Silhouette Score: ",score)
```

Output:

```
              precision    recall  f1-score   support

           0       0.00      1.00      0.00       0.0
           1       0.00      1.00      0.00       0.0
           2       0.00      1.00      0.00       0.0
           6       1.00      0.00      0.00     204.0
           7       1.00      0.00      0.00     190.0
           8       1.00      0.00      0.00     196.0
           9       1.00      0.00      0.00     175.0
          10       1.00      0.00      0.00     236.0
          11       1.00      0.00      0.00     365.0
          12       1.00      0.00      0.00    7584.0

    accuracy                           0.00    8950.0
   macro avg       0.70      0.30      0.00    8950.0
weighted avg       1.00      0.00      0.00    8950.0

[[   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [ 176   27    1    0    0    0    0    0    0    0]
 [ 171   17    2    0    0    0    0    0    0    0]
 [ 168   28    0    0    0    0    0    0    0    0]
 [ 149   26    0    0    0    0    0    0    0    0]
 [ 185   50    1    0    0    0    0    0    0    0]
 [ 276   86    3    0    0    0    0    0    0    0]
 [5184 2275  125    0    0    0    0    0    0    0]]

Accuracy for our Training dataset with PCA: 0.0
Silhouette Score:  0.5032491624765585
```

1.c Performing scaling + PCA + kmeans

Scaling the dataset

In [91]:

```
#Scaling the dataset
scaler = StandardScaler()
scaler.fit(x)
X_scaled_array = scaler.transform(x)
```

To apply PCA

```
In [93]: #applying PCA
pca = PCA(3)
x_pca = pca.fit_transform(X_scaled_array)
principalDf = pd.DataFrame(data = x_pca, columns = ['principal component 1', 'principal component 2','principal component 3'])
finalDf = pd.concat([principalDf, dataset.iloc[:,-1]], axis = 1)
finalDf.head()
```

Output:

Out[93]:

	principal component 1	principal component 2	principal component 3	TENURE
0	-1.588435	-1.021501	0.599838	12
1	-1.404531	2.278100	0.502689	12
2	0.929071	-0.572354	0.402758	12
3	-0.932885	-0.134715	1.748161	12
4	-1.563205	-0.774378	0.545853	12

Applying k-means and calculating silhouette score for train data

```
In [95]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.34,random_state=0)
nclusters = 3
# this is the k in kmeans
km = KMeans(n_clusters=nclusters)
km.fit(X_train,y_train)


# predict the cluster for each training data point
y_clus_train = km.predict(X_train)

# Summary of the predictions made by the classifier
print(classification_report(y_train, y_clus_train, zero_division=1))
print(confusion_matrix(y_train, y_clus_train))

train_accuracy = accuracy_score(y_train, y_clus_train)
print("Accuracy for our Training dataset with PCA:", train_accuracy)

#Calculate sihouette Score
score = metrics.silhouette_score(X_train, y_clus_train)
print("Sihouette Score: ",score)
```

Output:

```
              precision    recall  f1-score   support

           0       0.00      1.00      0.00       0.0
           1       0.00      1.00      0.00       0.0
           2       0.00      1.00      0.00       0.0
           6       1.00      0.00      0.00     139.0
           7       1.00      0.00      0.00     135.0
           8       1.00      0.00      0.00     128.0
           9       1.00      0.00      0.00     118.0
          10       1.00      0.00      0.00     151.0
          11       1.00      0.00      0.00     262.0
          12       1.00      0.00      0.00    4974.0

    accuracy                           0.00    5907.0
   macro avg       0.70      0.30      0.00    5907.0
weighted avg       1.00      0.00      0.00    5907.0

[[   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [  57   81    1    0    0    0    0    0    0    0]
 [  50   85    0    0    0    0    0    0    0    0]
 [  48   79    1    0    0    0    0    0    0    0]
 [  49   68    1    0    0    0    0    0    0    0]
 [  48  102    1    0    0    0    0    0    0    0]
 [  90  167    5    0    0    0    0    0    0    0]
 [2134 2514  326    0    0    0    0    0    0    0]]
Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.35908319934353417
```

Applying k-means and calculating silhouette score for test data

```
In [96]: # predict the cluster for each testing data point
         y_clus_test = km.predict(X_test)

         # Summary of the predictions made by the classifier
         print(classification_report(y_test, y_clus_test, zero_division=1))
         print(confusion_matrix(y_test, y_clus_test))

         train_accuracy = accuracy_score(y_test, y_clus_test)
         print("\nAccuracy for our Training dataset with PCA:", train_accuracy)

         #Calculate sihouette Score
         score = metrics.silhouette_score(X_test, y_clus_test)
         print("Sihouette Score: ",score)
```

Output:

```
              precision    recall  f1-score   support

           0       0.00      1.00      0.00       0.0
           1       0.00      1.00      0.00       0.0
           2       0.00      1.00      0.00       0.0
           6       1.00      0.00      0.00      65.0
           7       1.00      0.00      0.00      55.0
           8       1.00      0.00      0.00      68.0
           9       1.00      0.00      0.00      57.0
          10       1.00      0.00      0.00      85.0
          11       1.00      0.00      0.00     103.0
          12       1.00      0.00      0.00    2610.0

    accuracy                           0.00    3043.0
   macro avg       0.70      0.30      0.00    3043.0
weighted avg       1.00      0.00      0.00    3043.0

[[   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [  21   43    1    0    0    0    0    0    0    0]
 [  23   31    1    0    0    0    0    0    0    0]
 [  21   47    0    0    0    0    0    0    0    0]
 [  15   42    0    0    0    0    0    0    0    0]
 [  31   52    2    0    0    0    0    0    0    0]
 [  33   69    1    0    0    0    0    0    0    0]
 [1178 1254  178    0    0    0    0    0    0    0]]

Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.35017641810997635
```

2.a Scaling the data

```
In [102]: #2.a perform scaling on the dataset
          scaler = StandardScaler()
          X_Scale = scaler.fit_transform(X)
```

2.b Apply PCA with k=3

```
In [103]: #2.b apply PCA with k = 3
          pca3 = PCA(n_components=3)
          principalComponents = pca3.fit_transform(X_Scale)

          principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2','Principal Co

          finalDf = pd.concat([principalDf, dataset_pd[['class']]], axis = 1)
          finalDf.head()
```

Output:

Out[103]:

	principal component 1	principal component 2	Principal Component 3	class
0	-10.047372	1.471076	-6.846402	1
1	-10.637725	1.583749	-6.830976	1
2	-13.516185	-1.253542	-6.818696	1
3	-9.155083	8.833599	15.290906	1
4	-6.764470	4.611467	15.637122	1

2.c To perform SVM

```
In [105]: #2.c to perform svm and generate the report

          from sklearn.svm import SVC

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

          y_pred = svmClassifier.predict(X_test)

          # Summary of the predictions made by the classifier
          print(classification_report(y_test, y_pred, zero_division=1))
          print(confusion_matrix(y_test, y_pred))
          # Accuracy score
          glass_acc_svc = accuracy_score(y_pred,y_test)
          print('accuracy is',glass_acc_svc )

          #Calculate sihouette Score
          score = metrics.silhouette_score(X_test, y_pred)
          print("Sihouette Score: ",score)
```

Report of SVM

Output:

```
              precision    recall  f1-score   support

           0       0.67      0.42      0.51        62
           1       0.84      0.93      0.88       196

    accuracy                           0.81       258
   macro avg       0.75      0.68      0.70       258
weighted avg       0.80      0.81      0.79       258

[[ 26  36]
 [ 13 183]]
accuracy is 0.810077519379845
Sihouette Score:  0.2504463997042778
```

3. To apply LDA on iris dataset to reduce dimensionality of data to k = 2

Finding initial dimension

In [108]:
```
x = dataset_iris.iloc[:,1:-1]
y = dataset_iris.iloc[:,-1]
print(x.shape,y.shape)
```

Output:

```
(150, 4) (150,)
```

After applying LDA and dimension

In [111]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=2)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
print(X_train.shape,X_test.shape)
```

Output:

```
(105, 2) (45, 2)
```

4. Difference between PCA and LDA

PCA reduces features into orthogonal variables called principal components. The first one contains largest variability of data and it decreases for the next. LDA minimizes the variance within class and maximizes variance between categories.

They both are linear transformations which aim to maximize the variance in a lower dimension.

PCA is unsupervised and LDA is supervised learning algorithm. PCA finds directions of maximum variance regardless of class labels where LDA finds directions of maximum class separability.