Assignment- 5 ML

Nirmala Yarlagadda

700733102

Programming elements:

Principal Component Analysis In class programming:

1. Principal Component Analysis

a. Apply PCA on CC dataset.

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

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

Assignment-5

Question 1

1. Principal Component Analysis

In [168]:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, accuracy_score
from sklearn.decomposition import PCA
```

a. Apply PCA on CC dataset.

In [169]:
```
cc = pd.read_csv("C:/Users/Dell/Desktop/ML5/datasets5/datasets/CC.csv")
cc.head()
```

Out[169]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083

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

In [170]:
```
cc.describe()
```

Out[170]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0.490351
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371
min	0.000000	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	0.083333
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000

In [171]:
```
# delete CUST_ID
cc.drop(["CUST_ID"], axis=1, inplace=True)
```

In [172]:
```
cc.describe()
#cc.hist(bins=30)
#cc.boxplot() # remove any outliers if found
```

Out[172]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0.490351
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371
min	0.000000	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	0.083333
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000

In [173]:
```
#fill missing values in MINIMUM_PAYMENTS
mean_min_pay = int(cc.MINIMUM_PAYMENTS.dropna().mean())
cc['MINIMUM_PAYMENTS'] = cc['MINIMUM_PAYMENTS'].fillna(mean_min_pay)
```

In [174]:
```
#fill missing values in CREDIT_LIMIT
mean_cred_lim = int(cc.CREDIT_LIMIT.dropna().mean())
cc['CREDIT_LIMIT'] = cc['CREDIT_LIMIT'].fillna(mean_cred_lim)
```

In [175]:
```
#elbow is at k=5
km = KMeans(n_clusters=5, random_state=0)
km.fit_predict(cc)
score = silhouette_score(cc, km.labels_, metric='euclidean')
print('Initial Silhouetter Score: %.3f' % score)
```

```
Initial Silhouetter Score: 0.379
```

In [176]:
```
pca = PCA(n_components=5)
pca.fit(cc)
cc_pca = pd.DataFrame(pca.transform(cc), columns = ['A', 'B', 'C', 'D', 'E'])
```

In [177]:
```
#elbow is at k=5
km = KMeans(n_clusters=5, random_state=0)
km.fit_predict(cc_pca)
score = silhouette_score(cc_pca, km.labels_, metric='euclidean')
print('PCA Silhouetter Score: %.3f' % score)
```

```
PCA Silhouetter Score: 0.416
```

In [178]:
```
mms = MinMaxScaler()
mms.fit(cc)
cc_transformed_mms = mms.transform(cc)
```

2. Use pd_speech_features.csv

a. Perform Scaling

b. Apply PCA (k=3)

c. Use SVM to report performance

In [179]:
```
pca = PCA(n_components=5)
pca.fit(cc_transformed_mms)
cc_pca_transformed = pd.DataFrame(pca.transform(cc_transformed_mms), columns = ['A', 'B', 'C', 'D', 'E'])
```

In [180]:
```
#elbow is at k=5
km = KMeans(n_clusters=5, random_state=0)
km.fit_predict(cc_pca_transformed)
score = silhouette_score(cc_pca_transformed, km.labels_, metric='euclidean')
print('Scaled PCA Silhouetter Score: %.3f' % score)
```

```
Scaled PCA Silhouetter Score: 0.383
```

Question 2

2. Use pd_speech_features.csv

In [181]:
```
speech_f = pd.read_csv("C:/Users/Dell/Desktop/ML5/datasets5/datasets//pd_speech_features.csv")
speech_f.head()
```

Out[181]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kurtosisValue_dec_28	tq
0	0	1	0.85247	0.71826	0.57227	240	239	0.008064	0.000087	0.00218	...	1.5620	
1	0	1	0.76686	0.69481	0.53966	234	233	0.008258	0.000073	0.00195	...	1.5589	
2	0	1	0.85083	0.67604	0.58982	232	231	0.008340	0.000060	0.00176	...	1.5643	
3	1	0	0.41121	0.79672	0.59257	178	177	0.010858	0.000183	0.00419	...	3.7805	
4	1	0	0.32790	0.79782	0.53028	236	235	0.008162	0.002669	0.00535	...	6.1727	

5 rows × 755 columns

In [182]:
```
speech_f.describe()
```

Out[182]:

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqw
count	756.000000	756.000000	756.000000	756.000000	756.000000	756.000000	756.000000	756.000000	756.000000	756.000000	...	
mean	125.500000	0.515873	0.746284	0.700414	0.489058	323.972222	322.678571	0.006360	0.000383	0.002324	...	
std	72.793721	0.500079	0.169294	0.069718	0.137442	99.219059	99.402499	0.001826	0.000728	0.002628	...	
min	0.000000	0.000000	0.041551	0.543500	0.154300	2.000000	1.000000	0.002107	0.000011	0.000210	...	
25%	62.750000	0.000000	0.762833	0.647053	0.386537	251.000000	250.000000	0.005003	0.000049	0.000970	...	
50%	125.500000	1.000000	0.809655	0.700525	0.484355	317.000000	316.000000	0.006048	0.000077	0.001495	...	
75%	188.250000	1.000000	0.834315	0.754985	0.586515	384.250000	383.250000	0.007528	0.000171	0.002520	...	
max	251.000000	1.000000	0.907660	0.852640	0.871230	907.000000	905.000000	0.012966	0.003483	0.027750	...	

8 rows × 755 columns

a. Perform Scaling

In [183]:
```
# delete CUST_ID
speech_f.drop(["id"], axis=1, inplace=True)
```

In [184]:
```
x_speech = speech_f.drop(['class'], axis=1)
y_speech = speech_f.loc[:,'class']
```

In [185]:
```
from sklearn import svm
clf = svm.SVC()
clf.fit(x_speech, y_speech)
y_pred=clf.predict(x_speech)
print(accuracy_score(y_speech, y_pred))
```

```
0.7566137566137566
```

b. Apply PCA (k=3)

In [186]:
```
mms = MinMaxScaler()
mms.fit(x_speech)
x_speech_transformed_mms = mms.transform(x_speech)
```

In [187]:
```
clf = svm.SVC()
clf.fit(x_speech_transformed_mms, y_speech)
y_pred=clf.predict(x_speech_transformed_mms)
print(accuracy_score(y_speech, y_pred))
```

```
0.8703703703703703
```

c. Use SVM to report performance

In [188]:
```
pca = PCA(n_components=100)
pca.fit(x_speech_transformed_mms)
speech_pca_transformed = pd.DataFrame(pca.transform(x_speech_transformed_mms))#, columns = ['A', 'B', 'C']
```

In [189]:
```
clf = svm.SVC()
clf.fit(speech_pca_transformed, y_speech)
y_pred=clf.predict(speech_pca_transformed)
print(accuracy_score(y_speech, y_pred))
```

```
0.9325396825396826
```

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

Question 3

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

In [190]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

In [191]:
```
iris=pd.read_csv("C:/Users/Dell/Desktop/ML5/datasets5/datasets//Iris.csv")
iris.head()
```

Out[191]:

	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

In [192]:
```
iris.describe()
```

Out[192]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [193]:
```
iris_x = iris.drop(["Species"], axis=1)
iris_y = iris.loc[:,"Species"]
```

In [194]:
```
lda = LDA(n_components=2)
iris_x_transformed = lda.fit(iris_x, iris_y).transform(iris_x)
```

In [139]:
```
print(iris_x_transformed)
```

```
[[-1.00367633e+01 -4.51330244e-01]
 [-9.17292994e+00 -1.47723373e+00]
 [-9.48098912e+00 -9.79692560e-01]
 [-8.81811924e+00 -1.40860220e+00]
 [-9.96020031e+00 -1.12546395e-01]
 [-9.52340255e+00  4.51643380e-01]
 [-9.09952354e+00 -4.86482040e-01]
 [-9.36783890e+00 -5.04697982e-01]
```

4. Briefly identify the difference between PCA and LDA

Principal Component Analysis (PCA):

The way Principal Component Analysis (PCA) functions is by locating the directions (components) in a dataset that maximize the variance. In other words, it looks for the linear feature combination that captures the most variance. The largest variance is captured by the first component, which is orthogonal to the second and captures the remaining volatility, and so on. When your data shows linear correlations between features, or when you can define one feature as a function of another, PCA is a good technique for dimensionality reduction (s). By selecting the ideal amount of features, you can use PCA to compress your data while preserving most of the information content in such circumstances (components).

Linear discriminant analysis (LDA):

A further method of linear transformation used to reduce the dimensionality is linear discriminant analysis (LDA). LDA is a supervised learning approach, in contrast to PCA, and as such, when determining the directions of maximum variance, it considers class labels. Since you want to maximize class separability, LDA is especially well-suited for classification jobs.LDA makes the same assumptions as PCA regarding the origin of your data and the unidirectional nature of your features. Utilizing the Linear Discriminant Analysis and StandardScaler classes from scikit-learn, you can both center and decorrelate your data. Using the fit transform() method of scikit-learn after your data has been cleaned and transformed, you may fit an LDA model to it.

Video link:
https://drive.google.com/file/d/1mFfwm6imFGdijQqp7uDOZ_z5zCwvwvP5/view?usp=share_link

GitHub link: https://github.com/niryarjessy22/Assignment-5.git