Assignment 5 Documentation

Kavyasri

700728990

1. CC Dataset---PCA---K Means Clustering

- ✓ Read cc dataset using pandas.
- ✓ Check head to see columns and type of data.
- ✓ Drop CUST_ID column.
- ✓ Check for null values using isnull().sum().
- ✓ Fill missing values with column means using fillna() method.
- ✓ Use the k=5 to fit, predict KMeans on the data.
- ✓ Calculate silhouette_score. In our case its 0.379 for k=5.
- ✓ Now use n_components=5 to fit, transform cc data with PCA.
- ✓ Use the k=5 to fit, predict KMeans on the transformed data.
- ✓ Calculate silhouette_score. In our case its 0.416 for k=5.
- ✓ Now do MinMaxScaling on cc data to bring uniformity to various columns. Then do PCA on scaled data with same configuration above.
- ✓ Use the k=5 to fit, predict KMeans on the scale transformed data.
- ✓ Calculate silhouette_score. In our case its 0.383 for k=5.

Initial KMeans fit on original data:

```
In [9]: #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
```

KMeans fit on PCA transformed data:

```
In [10]: pca = PCA(n_components=5)
         pca.fit(cc)
         cc_pca = pd.DataFrame(pca.transform(cc), columns = ['A', 'B', 'C', 'D', 'E'])

In [11]: #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
```

Using PCA has improved clustering silhouette score.

KMeans fit on MinMax scaled PCA transformed data:

```
In [12]:
mms = MinMaxScaler()
mms.fit(cc)
cc_transformed_mms = mms.transform(cc)

In [13]:
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 [14]:
#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
```

2. Speech Dataset---PCA---SVM

- ✓ Read speech dataset using pandas.
- ✓ Check head to see columns and type of data.
- ✓ Drop “id” column as it is not useful for modeling.
- ✓ Check for null values using isnull().sum(). No missing values found.
- ✓ Separate X and Y variables with Y as “class”.
- ✓ Fit, predict SVM using this X and Y. Calculate accuracy. Its 0.756.
- ✓ Do MinMax scaling and Fit, predict SVM using this scaled_X and Y. Calculate accuracy. Its 0.87.
- ✓ Now Perform PCA on scaled data. Use this to fit SVM again and calculate accuracy.
- ✓ With n_components=3,5,10,30,50,100 test for accuracies. **We observe an increase in accuracy as n_components increase. As small number of components doesn’t hold(represent) all the required information of variables.**

Initial SVM accuracy:

```
In [20]:
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
```

Accuracy of SVM on scaled data:

In [21]:
```
mms = MinMaxScaler()
mms.fit(x_speech)
x_speech_transformed_mms = mms.transform(x_speech)
```

In [22]:
```
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
```

Accuracy of SVM on scaled PCA data: (n_components=3,5,10,30,50,100)

In [39]:
```
pca = PCA(n_components=3)
pca.fit(x_speech_transformed_mms)
speech_pca_transformed = pd.DataFrame(pca.transform(x_speech_transformed_mms))#
```

In [40]:
```
clf = svm.SVC()
clf.fit(speech_pca_transformed, y_speech)
y_pred=clf.predict(speech_pca_transformed)
print(accuracy_score(y_speech, y_pred))
```

```
0.8042328042328042
```

In [41]:
```
pca = PCA(n_components=5)
pca.fit(x_speech_transformed_mms)
speech_pca_transformed = pd.DataFrame(pca.transform(x_speech_transformed_mms))
```

In [42]:
```
clf = svm.SVC()
clf.fit(speech_pca_transformed, y_speech)
y_pred=clf.predict(speech_pca_transformed)
print(accuracy_score(y_speech, y_pred))
```

```
0.8359788359788359
```

In [43]:
```
pca = PCA(n_components=10)
pca.fit(x_speech_transformed_mms)
speech_pca_transformed = pd.DataFrame(pca.transform(x_speech_transformed_mms))
```

In [44]:
```
clf = svm.SVC()
clf.fit(speech_pca_transformed, y_speech)
y_pred=clf.predict(speech_pca_transformed)
print(accuracy_score(y_speech, y_pred))
```

```
0.8664021164021164
```

In [45]:
```
pca = PCA(n_components=30)
pca.fit(x_speech_transformed_mms)
speech_pca_transformed = pd.DataFrame(pca.transform(x_speech_transformed_mms))
```

In [46]:
```
clf = svm.SVC()
clf.fit(speech_pca_transformed, y_speech)
y_pred=clf.predict(speech_pca_transformed)
print(accuracy_score(y_speech, y_pred))
```

```
0.9074074074074074
```

In [47]:
```
pca = PCA(n_components=50)
pca.fit(x_speech_transformed_mms)
speech_pca_transformed = pd.DataFrame(pca.transform(x_speech_transformed_mms))
```

In [48]:
```
clf = svm.SVC()
clf.fit(speech_pca_transformed, y_speech)
y_pred=clf.predict(speech_pca_transformed)
print(accuracy_score(y_speech, y_pred))
```

```
0.9206349206349206
```

In [49]:
```
pca = PCA(n_components=100)
pca.fit(x_speech_transformed_mms)
speech_pca_transformed = pd.DataFrame(pca.transform(x_speech_transformed_mms))
```

In [50]:
```
clf = svm.SVC()
clf.fit(speech_pca_transformed, y_speech)
y_pred=clf.predict(speech_pca_transformed)
print(accuracy_score(y_speech, y_pred))
```

```
0.9298941798941799
```

3. IRIS Dataset---LDA

- ✓ Read iris dataset using pandas.
- ✓ Check head to see columns and type of data.
- ✓ Separate X and Y variables with Y as “Species”.
- ✓ Use LDA from sklearn to transform the data with n_components=2.
- ✓ Check for the transformed variables.

```
In [28]: iris_x = iris.drop(["Species"], axis=1)
         iris_y = iris.loc[:,"Species"]

In [29]: lda = LDA(n_components=2)
         iris_x_transformed = lda.fit(iris_x, iris_y).transform(iris_x)

In [30]: print(iris_x_transformed)
```

```
[-9.96642412e+00  1.39824256e+00]
[-9.19297836e+00  3.48117706e-01]
[-9.36227503e+00  8.31991272e-01]
[-9.32391418e+00  9.16315048e-01]
[-8.76248233e+00 -5.62718432e-02]
[-8.88178087e+00  8.74105504e-01]
[-9.87055894e+00  1.00209591e+00]
[-7.60804498e+00 -8.20642233e-03]
[-7.77791864e+00 -3.20242776e-01]
[-7.96141242e+00 -6.57043051e-01]
[-8.00979719e+00  3.34031714e-01]
[-8.92701582e+00  6.16194234e-01]
[-8.96319773e+00  5.77438977e-01]
[-7.88178440e+00 -1.83196764e-01]
[-7.77863981e+00 -3.08300446e-01]
[-8.32075648e+00  7.97946692e-01]
[-9.80491422e+00  1.96064097e+00]
[-1.00940635e+01  2.49438691e+00]
[-8.14751552e+00 -5.83887141e-02]
```

3. PCA vs LDA

- ✓ **While PCA and LDA are both dimensionality reduction techniques, LDA attempts to find a feature subspace that maximizes class separability and PCA has no such constraints. So in LDA we have also used y variable while fitting the data.**