Task4

Build a classification model that can effectively analyze and extract features from an image. Apply PCA algorithm to find the appropriate feature.

Tools: Rapid Miner, Python, Scikitlearn, Anaconda navigator

Exp. No.: 4

Build a classification model that can effectively analyze and extract features from an image. Apply PCA algorithm to find the appropriate feature.

Algorithm

Step 1: Standardize the dataset.

Step 2: Calculate the covariance matrix for the features in the dataset.

Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix.

Step 4: Sort eigenvalues and their corresponding eigenvectors.

Step 5: Pick k eigenvalues and form a matrix of eigenvectors.

Step 6: Transform the original matrix.

Code:

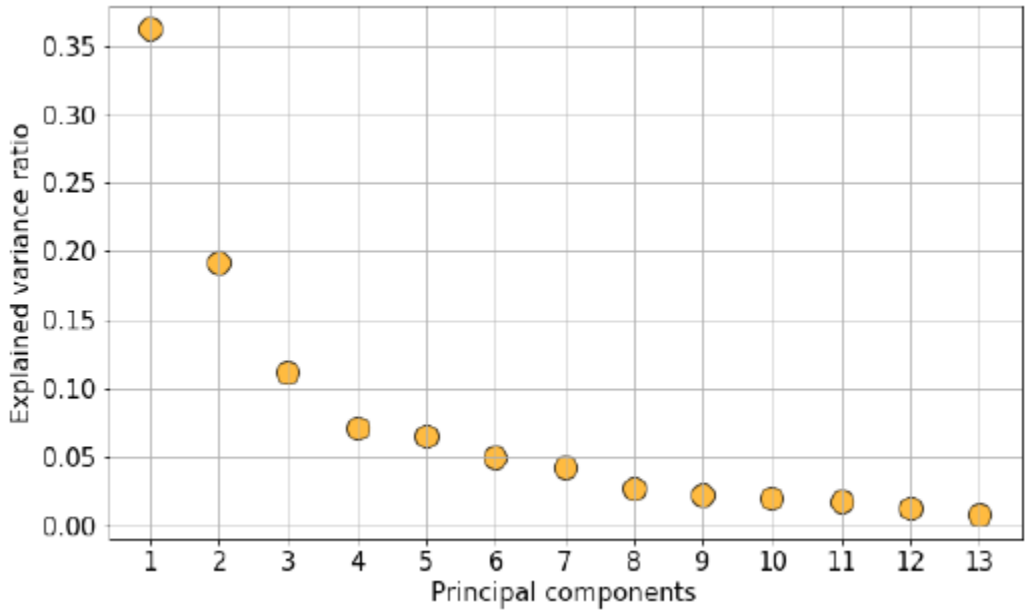
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
df = pd.read_csv('wine.data.csv')
df.head(10)
df.iloc[:,1:].describe()
```

```
for c in df.columns[1:]:
df.boxplot(c,by='Class',figsize=(7,4),fontsize=14)
plt.title("{}\n".format(c),fontsize=16)
plt.xlabel("Wine Class", fontsize=16)
plt.figure(figsize=(10,6))
plt.scatter(df['OD280/OD315 of diluted
wines'],df['Flavanoids'],c=df['Class'],edgecolors='k',alpha=0.75,s=150)
plt.grid(True)
plt.title("Scatter plot of two features showing the \ncorrelation and class
seperation",fontsize=15)
plt.xlabel("OD280/OD315 of diluted wines",fontsize=15)
plt.ylabel("Flavanoids",fontsize=15)
plt.show()
def correlation_matrix(df):
from matplotlib import pyplot as plt
from matplotlib import cm as cm
fig = plt.figure(figsize=(16,12))
ax1 = fig.add_subplot(111)
cmap = cm.get_cmap('jet', 30)
cax = ax1.imshow(df.corr(), interpolation="nearest", cmap=cmap)
ax1.grid(True)
plt.title('Wine data set features correlation\n',fontsize=15)
labels=df.columns
ax1.set_xticklabels(labels,fontsize=9)
ax1.set_yticklabels(labels,fontsize=9)
# Add colorbar, make sure to specify tick locations to match desired ticklabels
fig.colorbar(cax, ticks=[0.1*i for i in range(-11,11)])
```

```
plt.show()
correlation_matrix(df)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = df.drop('Class',axis=1)
y = df['Class']
X = scaler.fit_transform(X)
dfx = pd.DataFrame(data=X,columns=df.columns[1:])
dfx.head(10)
dfx.describe()
from sklearn.decomposition import PCA
pca = PCA(n_components=None)
dfx_pca = pca.fit(dfx)
plt.figure(figsize=(10,6))
plt.scatter(x=[i+1 for i in range(len(dfx_pca.explained_variance_ratio_))],
y=dfx_pca.explained_variance_ratio_,
s=200, alpha=0.75,c='orange',edgecolor='k')
plt.grid(True)
plt.title("Explained variance ratio of the \nfitted principal component vector\n",fontsize=25)
plt.xlabel("Principal components",fontsize=15)
plt.xticks([i+1 for i in range(len(dfx_pca.explained_variance_ratio_))],fontsize=15)
plt.yticks(fontsize=15)
plt.ylabel("Explained variance ratio",fontsize=15)
plt.show()
dfx_trans = pca.transform(dfx)
dfx_trans = pd.DataFrame(data=dfx_trans)
```

```
dfx_trans.head(10)
plt.figure(figsize=(10,6))
plt.scatter(dfx_trans[0],dfx_trans[1],c=df['Class'],edgecolors='k',alpha=0.75,s=150
)
plt.grid(True)
plt.title("Class separation using first two principal components\n",fontsize=20)
plt.xlabel("Principal component-1",fontsize=15)
plt.ylabel("Principal component-2",fontsize=15)
plt.show()
```

Explained variance ration of the fitted principal component vector



Class separation using first two principal components