Assignment 1

Implement **Principal Component Analysis** Algorithm and use it to reduce dimensions of Iris Dataset. Consider the following instructions

(a) Plot the magnitude of eigenvalues in sorted order.

(b) Plot the reconstructed data points along with the class labels using 1 and 2 PCs for reconstruction.

(c) Classify the dimension reduced dataset using bayes classifier.

Step by step implementation of PCA

1. Load the dataset
2. Normalize the data-scaling using Z-score scalar
3. Calculate the covariance matrix X of data points
4. Sort the eigen vectors according to their eigen values in decending order. Choose first k eigen vectors and that will be the new k dimensions
5. Transform the original n dimensional data points into k dimensions

Import files

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

1) Load the dataset

```
import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
```

```
X = df[["sepal_length", "sepal_width", "petal_length", "petal_width"]]
Y = df[["species"]]
```

OUTPUT

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

2) Normalize the data-scaling using Z-score scalar

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

3) Calculate the covariance matrix X of data points

```
import numpy as np
features = X_scaled.T
covariance_matrix = np.cov(features)
covariance_matrix
```

```
array([[ 1.00671141, -0.11835884,  0.87760447,  0.82343066],
       [-0.11835884,  1.00671141, -0.43131554, -0.36858315],
       [ 0.87760447, -0.43131554,  1.00671141,  0.96932762],
       [ 0.82343066, -0.36858315,  0.96932762,  1.00671141]])
```

4) Find the Eigen values and vector

```
values, vectors = np.linalg.eig(covariance_matrix)
```

```
vectors
```

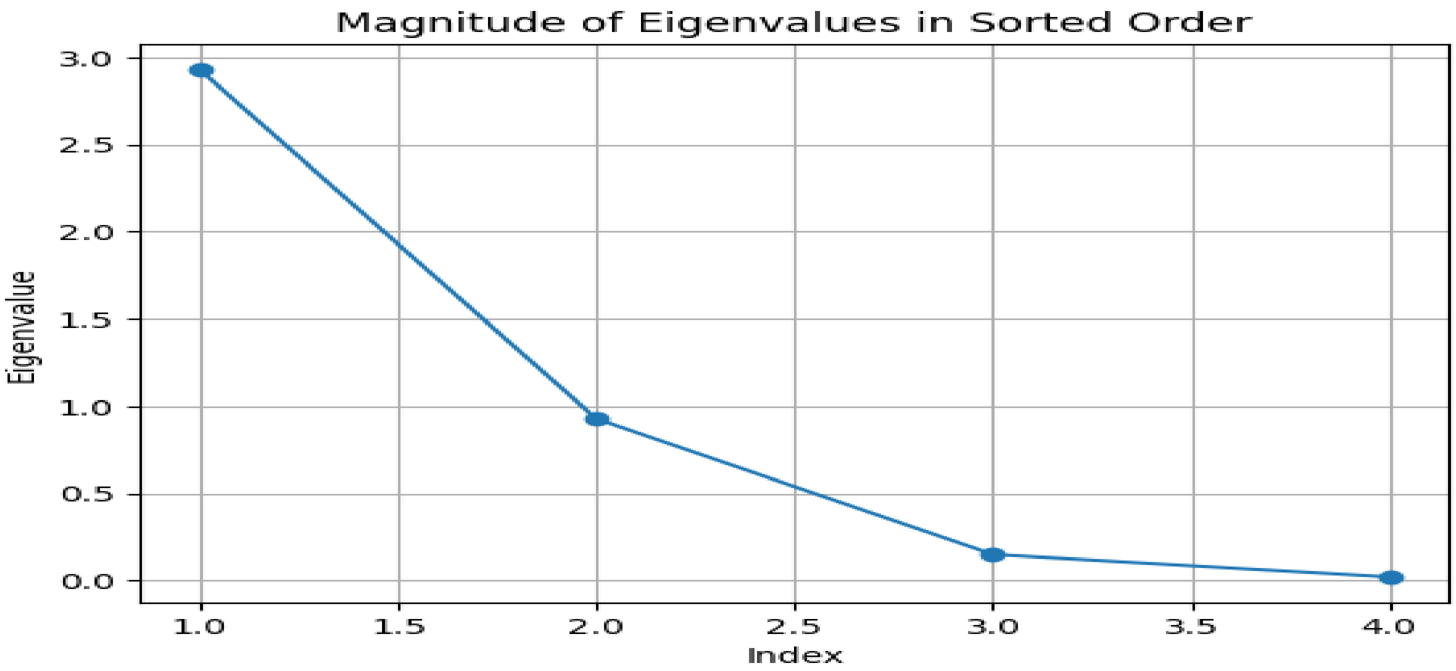
```
array([[ 0.52106591, -0.37741762, -0.71956635,  0.26128628],
       [-0.26934744, -0.92329566,  0.24438178, -0.12350962],
       [ 0.5804131 , -0.02449161,  0.14212637, -0.80144925],
       [ 0.56485654, -0.06694199,  0.63427274,  0.52359713]])
```

```
values
```

```
array([2.93808505, 0.9201649 , 0.14774182, 0.02085386])
```

(a) Plot the magnitude of eigenvalues in sorted order.

```
import matplotlib.pyplot as plt
sorted_indices = np.argsort(values)[::-1]
eigenvalues = values[sorted_indices]

# Plot the eigenvalues
plt.figure()
plt.plot(np.arange(1, len(values) + 1), values, 'o-')
plt.title('Magnitude of Eigenvalues in Sorted Order')
plt.xlabel('Index')
plt.ylabel('Eigenvalue')
plt.grid(True)
plt.show()
```



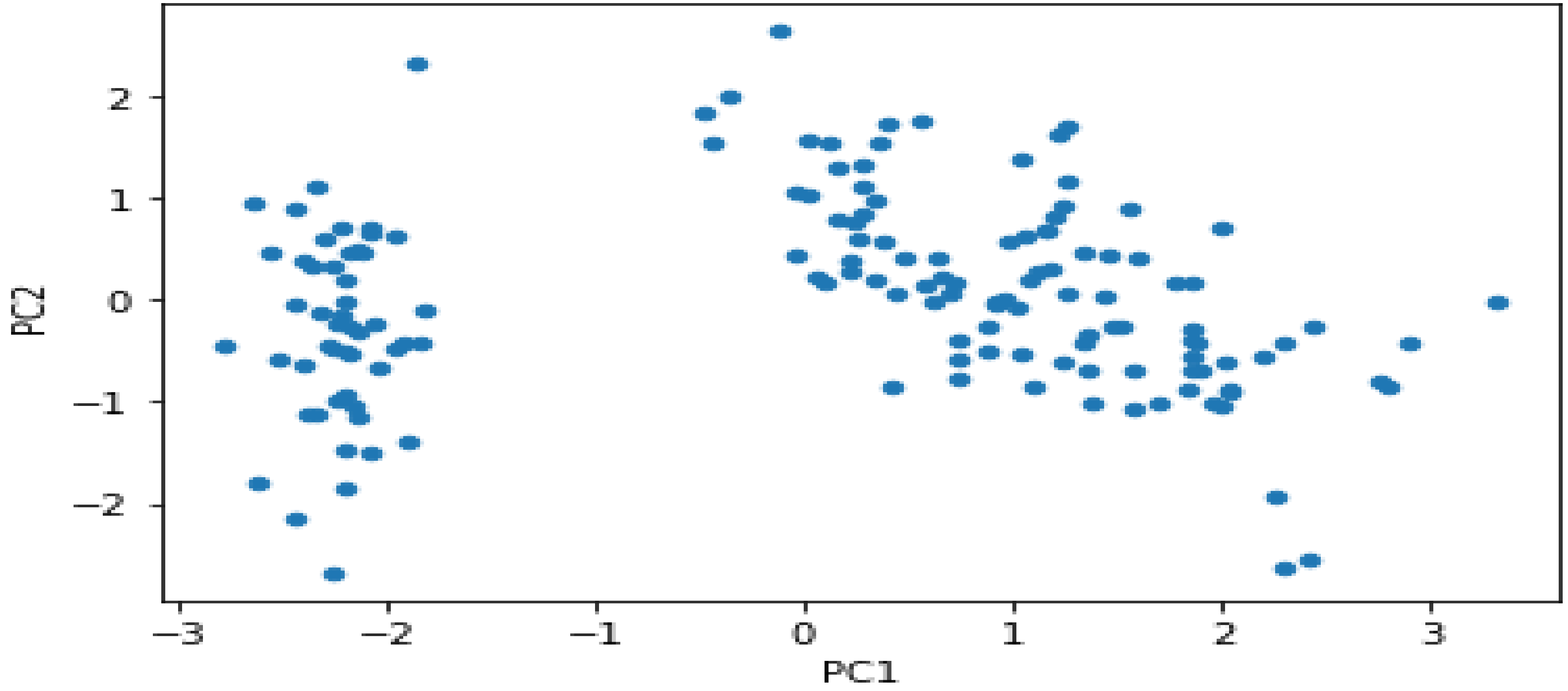
5) Choose first k eigen vectors and that will be the new k dimensions

```
projected_1 = X_scaled.dot(vectors.T[0])
projected_2 = X_scaled.dot(vectors.T[1])
result = pd.DataFrame(projected_1, columns = ['PC1'])
result['PC2'] = projected_2
result['species'] = Y
result
```

b)Plot the results

```
result.plot(kind = "scatter", x = 'PC1', y = 'PC2')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f6c02d79bd0>



c) Classify the dimension reduced dataset using bayes classifier

```
X_train, X_test, y_train, y_test = train_test_split(X_pda, Y, test_size=0.3, random_state=42)

# Fit a Gaussian Naive Bayes classifier
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Predict the class labels
y_pred = gnb.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))

# Plot the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
Accuracy: 0.89
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       0.90      0.69      0.78        13
   virginica       0.75      0.92      0.83        13

    accuracy                           0.89        45
   macro avg       0.88      0.87      0.87        45
weighted avg       0.90      0.89      0.89        45

[[19  0  0]
 [ 0  9  4]
 [ 0  1 12]]
```

Assignment 2

Implement Linear Discriminant Analysis Algorithm and use it to reduce dimensions of Iris Dataset.

Consider the following instructions

- (a) Plot the transformed dataset along with the corresponding class labels.
- (b) Use bayes classifier for classification

Implement Linear Discriminant Analysis (LDA)

1. **Load the iris dataset**
2. **Compute the mean vectors**
3. **Compute the within-class scatter matrix**
4. **Compute the between-class scatter matrix**
5. **Compute the eigenvalues and eigenvectors**
6. **Select the top eigenvectors**
7. **Transform the dataset**

1) Load the dataset

```
import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
```

```
X = df[["sepal_length", "sepal_width", "petal_length", "petal_width"]]
Y = df[["species"]]
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Find number of features and classes

```
n_features = X.shape[1]
class_labels = np.unique(y)
```

2) Compute the mean vectors

```python
#Step 2: Compute the mean vectors for each class
mean_vectors = []
for cls in class_labels:
    mean_vectors.append(np.mean(X[y == cls], axis=0))
print(mean_vectors)
```

```
[array([5.006, 3.428, 1.462, 0.246]), array([5.936, 2.77 , 4.26 , 1.326]), array([6.588, 2.974, 5.552, 2.026])]
```

3) Compute the within-class scatter matrix

```python
# Step 3: Compute the within-class scatter matrix
S_W = np.zeros((n_features, n_features))
for cls, mean_vec in zip(class_labels, mean_vectors):
    class_scatter = np.zeros((n_features, n_features))
    for row in X[y == cls]:
        row, mean_vec = row.reshape(n_features, 1), mean_vec.reshape(n_features, 1)
        class_scatter += (row - mean_vec).dot((row - mean_vec).T)
    S_W += class_scatter
print(S_W)
```

```
[[38.9562 13.63    24.6246   5.645 ]
 [13.63    16.962    8.1208   4.8084]
 [24.6246   8.1208 27.2226   6.2718]
 [ 5.645    4.8084   6.2718   6.1566]]
```

4) Compute the between-class scatter matrix

```
:
# Step 4: Compute the between-class scatter matrix
overall_mean = np.mean(X, axis=0).reshape(n_features, 1)
S_B = np.zeros((n_features, n_features))
for i, mean_vec in enumerate(mean_vectors):
    n = X[y == i, :].shape[0]
    mean_vec = mean_vec.reshape(n_features, 1)
    overall_mean = overall_mean.reshape(n_features, 1)
    S_B += n * (mean_vec - overall_mean).dot((mean_vec - overall_mean).T)
print(S_B)
```

```
[[ 63.21213333 -19.95266667 165.2484      71.27933333]
 [-19.95266667  11.34493333 -57.2396     -22.93266667]
 [165.2484     -57.2396     437.1028     186.774     ]
 [ 71.27933333 -22.93266667 186.774       80.41333333]]
```

5) Compute the eigenvalues and eigenvectors

```
# Step 5: Solve the eigenvalue problem for the matrix inv(S_W).dot(S_B)
eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
print(eig_vals)
```

```
[ 3.21919292e+01  2.85391043e-01 -1.50470066e-15 -8.76730986e-15]
```

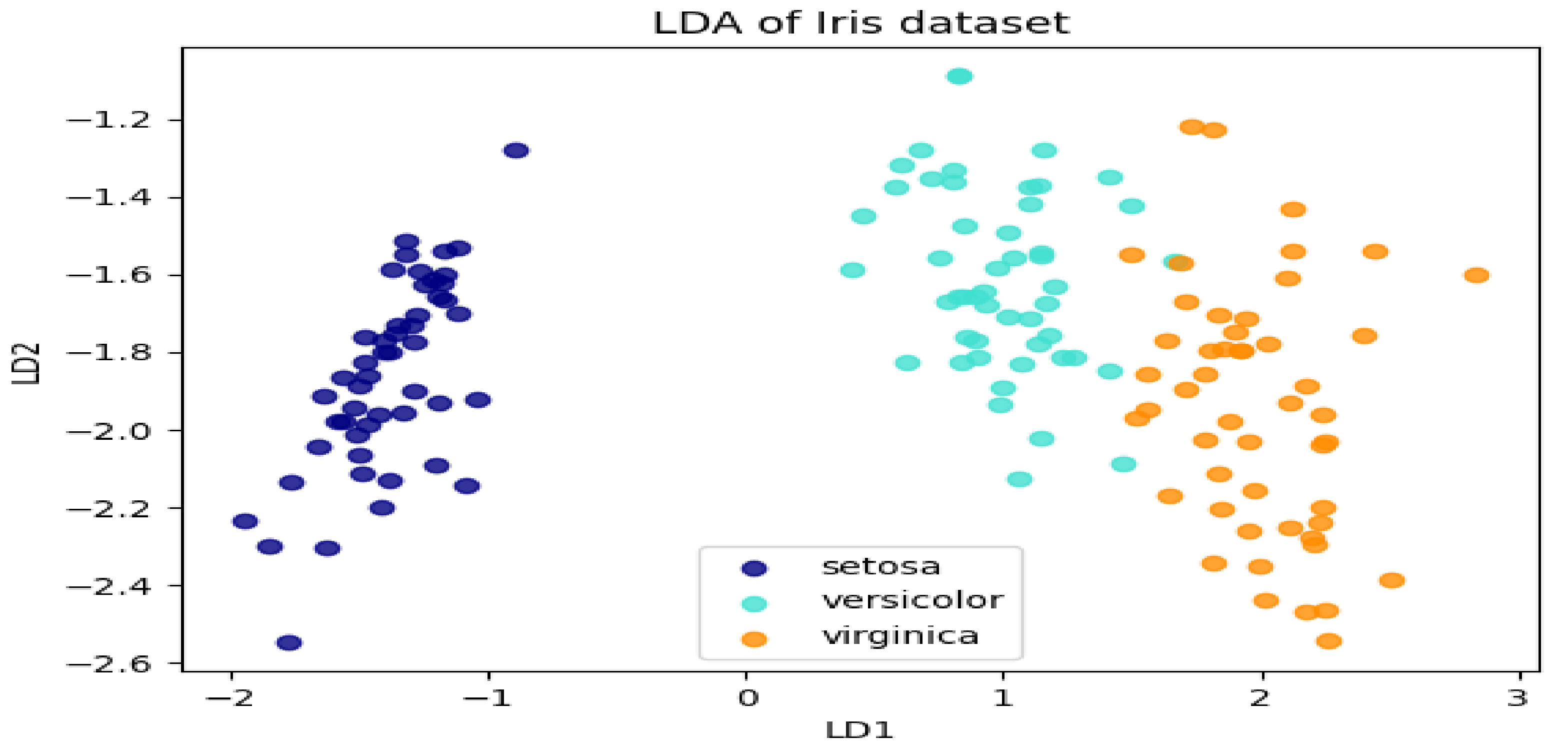
6) Select the top eigenvectors

```
# Step 6: Sort the eigenvalues and eigenvectors in descending order and Select the top eigenvectors

eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:, i]) for i in range(len(eig_vals))]
eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)
W = np.hstack((eig_pairs[0][1].reshape(n_features, 1), eig_pairs[1][1].reshape(n_features, 1)))
print(W)
```

```
[[-0.20874182 -0.00653196]
 [-0.38620369 -0.58661055]
 [ 0.55401172  0.25256154]
 [ 0.7073504  -0.76945309]]
```

a) Plot the transformed dataset along with the corresponding class labels.

```
# Step 7: Transform the dataset
X_lda = X.dot(W)

# Plot the LDA result
colors = ['navy', 'turquoise', 'darkorange']
plt.figure()
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_lda[y == i, 0], X_lda[y == i, 1], alpha=.8, color=color, label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA of Iris dataset')
plt.xlabel('LD1')
plt.ylabel('LD2')
plt.show()
```



b) Use bayes classifier for classification

```python
# Split the transformed dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_lda, y, test_size=0.3, random_state=42)

# Fit a Gaussian Naive Bayes classifier
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Predict the class labels
y_pred = gnb.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))

# Plot the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```