EEE418: Advanced Pattern Recognition **Spring 2020**

Lab 2: Feature Extraction

Student: Zhiyu Xu *ID: 1926906*

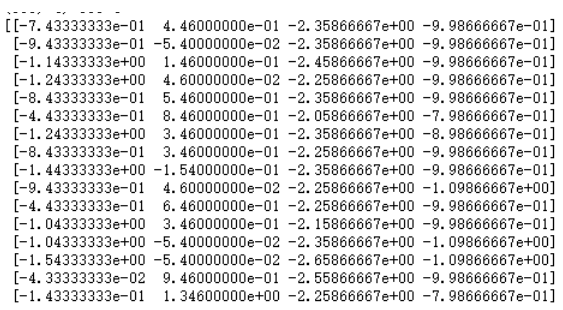
2.1 Principal Component Analysis (PCA)

1. Standardize the data $\widehat{X} = X_{n\times d} - 1_{n\times 1} mean(X)_{1\times d}$

   Code as shown below:

```
X = np.mat(iris_data)
X_mean = np.mean(X, axis=0)
X_meanRemoved = X    X_mean
```

   Result:



```
[[-7.43333333e-01  4.46000000e-01 -2.35866667e+00 -9.98666667e-01]
 [-9.43333333e-01 -5.40000000e-02 -2.35866667e+00 -9.98666667e-01]
 [-1.14333333e+00  1.46000000e-01 -2.45866667e+00 -9.98666667e-01]
 [-1.24333333e+00  4.60000000e-02 -2.25866667e+00 -9.98666667e-01]
 [-8.43333333e-01  5.46000000e-01 -2.35866667e+00 -9.98666667e-01]
 [-4.43333333e-01  8.46000000e-01 -2.05866667e+00 -7.98666667e-01]
 [-1.24333333e+00  3.46000000e-01 -2.35866667e+00 -8.98666667e-01]
 [-8.43333333e-01  3.46000000e-01 -2.25866667e+00 -9.98666667e-01]
 [-1.44333333e+00 -1.54000000e-01 -2.35866667e+00 -9.98666667e-01]
 [-9.43333333e-01  4.60000000e-02 -2.25866667e+00 -1.09866667e+00]
 [-4.43333333e-01  6.46000000e-01 -2.25866667e+00 -9.98666667e-01]
 [-1.04333333e+00  3.46000000e-01 -2.15866667e+00 -9.98666667e-01]
 [-1.04333333e+00 -5.40000000e-02 -2.35866667e+00 -1.09866667e+00]
 [-1.54333333e+00 -5.40000000e-02 -2.65866667e+00 -1.09866667e+00]
 [-4.33333333e-02  9.46000000e-01 -2.55866667e+00 -9.98666667e-01]
 [-1.43333333e-01  1.34600000e+00 -2.25866667e+00 -7.98666667e-01]
```

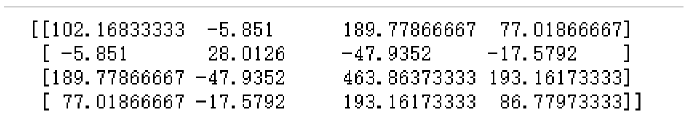
Figure 2.1: Standardize the datal

2. Compute the covariance matrix $\sum = \widehat{X}^T\widehat{X}$

   Code as shown below:

```
covMat = np.dot(X_meanRemoved.T, X_meanRemoved)
print(covMat)
```

   Result:



```
[[102.16833333  -5.851      189.77866667  77.01866667]
 [ -5.851       28.0126     -47.9352     -17.5792    ]
 [189.77866667 -47.9352     463.86373333 193.16173333]
 [ 77.01866667 -17.5792     193.16173333  86.77973333]]
```

Figure 2.2: covariance matrix

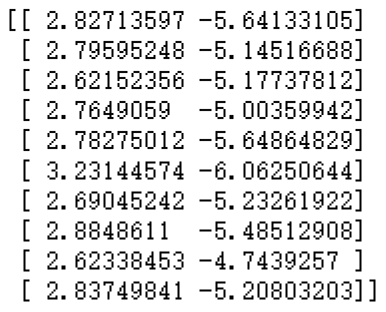
3. Transform the training dataset X using C

   Code as shown below:

```python
eigVals,eigVects=np.linalg.eig(np.mat(covMat))
eig_pairs = [(np.abs(eigVals[i]), eigVects[:,i]) for i in range(n_features)]
# sort eigVects based on eigVals from highest to lowest
eig_pairs.sort(reverse=True)
# Compute r eigenvectors with largest eigenvalues
C=np.array([ele[1] for ele in eig_pairs[:2]])
#get new data
Y=np.dot(X,np.transpose(C))
print(Y[:10])
```

Here we chose to reduce the original 4D data to 2D, so we selected the two largest feature values and their corresponding feature vectors.

Result:



```
[[ 2.82713597 -5.64133105]
 [ 2.79595248 -5.14516688]
 [ 2.62152356 -5.17737812]
 [ 2.7649059  -5.00359942]
 [ 2.78275012 -5.64864829]
 [ 3.23144574 -6.06250644]
 [ 2.69045242 -5.23261922]
 [ 2.8848611  -5.48512908]
 [ 2.62338453 -4.7439257 ]
 [ 2.83749841 -5.20803203]]
```

Figure 2.3: Top10 of Y_PCA

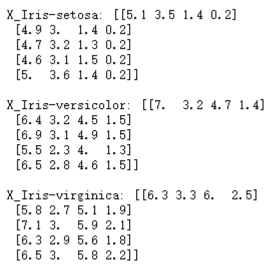
2.2 Linear Discriminant Analysis (LDA)

1. Generate $\{X_k\}$ matrix according to the labels of the examples.

   Code as shown below:

```python
X_K = []
for label,i in zip(['Iris setosa','Iris versicolor','Iris virginica'],range(n_features)):
        X_K.append(X[iris_target == label])
print('X_%s:%s\n' % (label, X_K[i][:5]))
```

Result:



```
X_Iris-setosa: [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]

X_Iris-versicolor: [[7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]]

X_Iris-virginica: [[6.3 3.3 6.  2.5]
 [5.8 2.7 5.1 1.9]
 [7.1 3.  5.9 2.1]
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]]
```

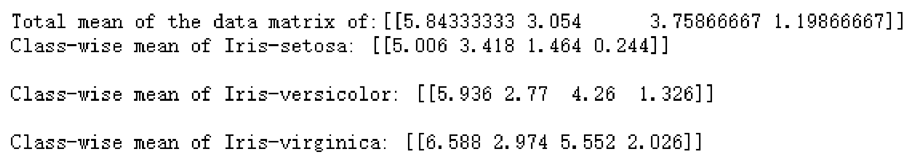
Figure 2.4: Top5 of each X_k

2. Compute the class-wise mean and the total mean of the data matrix

   Code as shown below:

```python
mean_class = []
mean_total = np.mean(X, axis=0)
print('Total␣mean␣of␣the␣data␣matrix␣of:%s'%mean_total)
for label,i in zip(['Iris setosa','Iris versicolor','Iris virginica'],range(n_features)):
        mean_class.append(np.mean(X_K[i], axis=0))
        print('Class wise␣mean␣of␣%s:␣%s\n' % (label, mean_class[i]))
```

   Result:



```
Total mean of the data matrix of:[[5.84333333 3.054      3.75866667 1.19866667]]
Class-wise mean of Iris-setosa: [[5.006 3.418 1.464 0.244]]

Class-wise mean of Iris-versicolor: [[5.936 2.77  4.26  1.326]]

Class-wise mean of Iris-virginica: [[6.588 2.974 5.552 2.026]]
```

Figure 2.5: Total mean and Class-vise mean

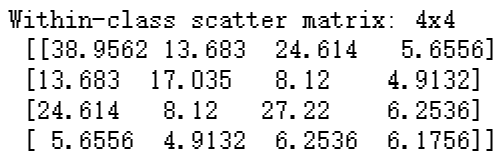
3. Compute the within-class covariance matrix S_W

   Code as shown below:

```python
d = n_features  # number of features
S_W = np.zeros((d, d))
for label, mc in zip(['Iris setosa','Iris versicolor','Iris virginica'], mean_class):
    class_scatter = np.zeros((d, d))  # scatter matrix for each class
    for row in X[iris_target == label]:
        row, mc = row.reshape(d, 1), mc.reshape(d, 1)  # make column vectors
        class_scatter += (row   mc).dot((row   mc).T)
        S_W += class_scatter                          # sum class scatter matrices

print('Within class␣scatter␣matrix:␣%sx%s␣\n' % (S_W.shape[0], S_W.shape[1]),S_W)
```

   Result:



```
Within-class scatter matrix: 4x4
 [[38.9562 13.683  24.614   5.6556]
 [13.683  17.035   8.12    4.9132]
 [24.614   8.12   27.22    6.2536]
 [ 5.6556  4.9132  6.2536  6.1756]]
```

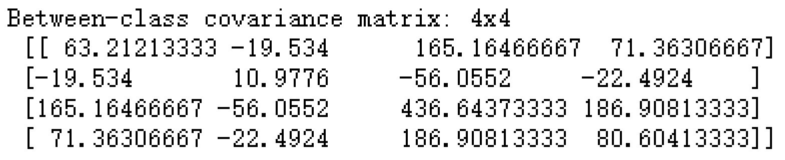
Figure 2.6: Within-class covariance matrix S_W

4. Compute the between-class covariance matrix S_B

   Code as shown below:

```python
S_B = np.zeros((d, d))
for label, mc in zip(['Iris setosa','Iris versicolor','Iris virginica'],mean_class):
        n = X[iris_target ==label,:].shape[0]
        mc = mc.reshape(d, 1)  # make column vector
        mean_total = mean_total.reshape(d, 1)  # make column vector
        S_B += n * (mc   mean_total).dot((mc   mean_total).T)
print(S_B))
```

Result:



```
Between-class covariance matrix: 4x4
 [[ 63.21213333 -19.534      165.16466667  71.36306667]
 [-19.534       10.9776     -56.0552     -22.4924    ]
 [165.16466667 -56.0552     436.64373333 186.90813333]
 [ 71.36306667 -22.4924     186.90813333  80.60413333]]
```

Figure 2.7: Betwenn-class covariance matrix S_B

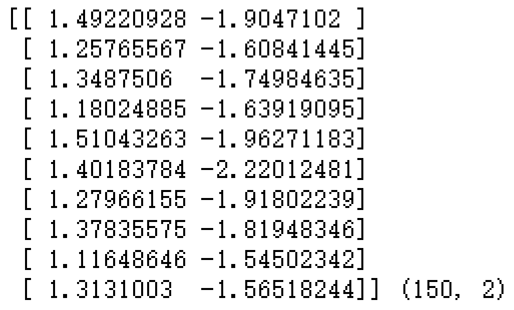
5. Find K-1 projection eigvector of $S_W^{-1}S_B$ as the matrix C and transform the data matrix X

   Code as shown below:

```
eigVals_LDA, eigVecs_LDA = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
eig_pairs_LDA = [(np.abs(eigVals_LDA[i]), eigVecs_LDA[:,i]) for i in range(n_features)]
# sort eig_vec based on eig_val from highest to lowest
eig_pairs_LDA.sort(reverse=True)
# Compute r eigenvectors with largest eigenvalues
C_LDAnp.array([ele[1] for ele in eig_pairs_LDA[:2]])
#get new data
Y_LDA=np.dot(X,np.transpose(C_LDA))
print(Y_LDA[:10],Y_LDA.shape)
```

   Here we chose to reduce the original 4D data to 2D, so we selected the two largest feature values and their corresponding feature vectors.

   Result:



```
[[ 1.49220928 -1.9047102 ]
 [ 1.25765567 -1.60841445]
 [ 1.3487506  -1.74984635]
 [ 1.18024885 -1.63919095]
 [ 1.51043263 -1.96271183]
 [ 1.40183784 -2.22012481]
 [ 1.27966155 -1.91802239]
 [ 1.37835575 -1.81948346]
 [ 1.11648646 -1.54502342]
 [ 1.3131003  -1.56518244]] (150, 2)
```

Figure 2.8: Top10 of Y_LDA

2.3 Comparisons of PCA and LDA

1. Project the dataset into 2 dimensional space and visualize them

   Code as shown below:

```
plt.figure(figsize=(16,9))
plt.subplot(1, 2, 1)
for color,target_name in zip("rgb", ['Iris setosa',
'Iris versicolor','Iris virginica']):
        plt.scatter(np.array(Y[iris_target == target_name, 0]),
        np.array(Y[iris_target == target_name, 1]),
        c=color, label=target_name)
plt.xlabel('Dimension1')
plt.ylabel('Dimension2')
plt.title("Iris_PCA")
plt.legend()

plt.subplot(1, 2, 2)
for color,target_name in zip("rgb", ['Iris setosa',
'Iris versicolor','Iris virginica']):
        plt.scatter(np.array(Y_LDA[iris_target == target_name, 0]),
        np.array(Y_LDA[iris_target == target_name, 1]),
        c=color, label=target_name)
plt.xlabel('Dimension1')
plt.ylabel('Dimension2')
plt.title("Iris_LDA")
plt.legend()
plt.show()
```

Result:

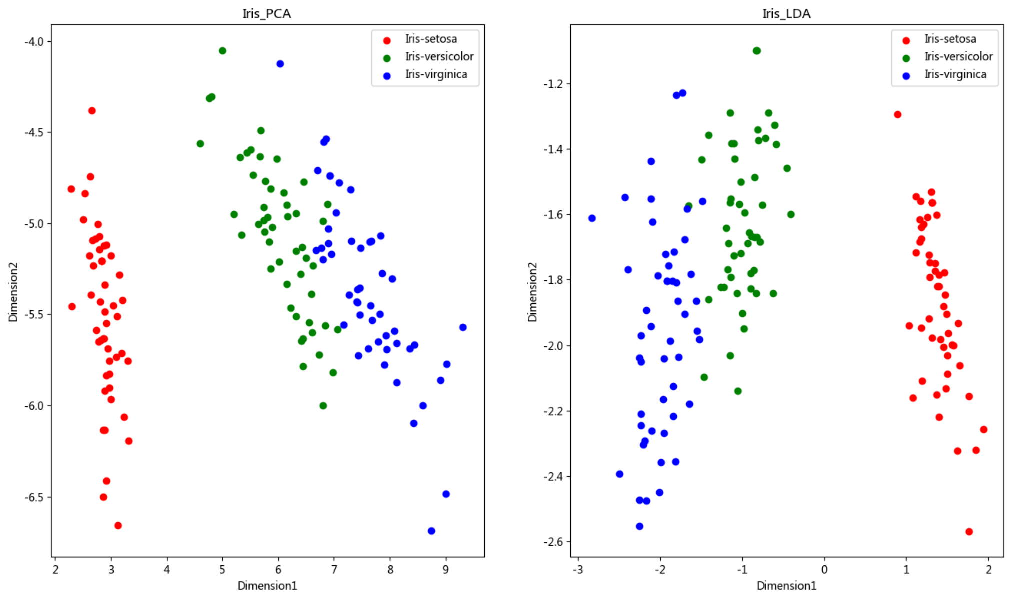


Figure 2.9: PCA and LDA

2. Compare the visualization of both algorithm and analyze their differences

   In order to verify the accuracy of the program, the PCA and LDA functions are used in the sklearn library to perform dimensionality reduction operations on the data. Code as shown below:

```
def PCA(data, n):
        from sklearn.decomposition import PCA
        pca = PCA(n_components=n)
        pca_result  = pca.fit_transform(data)
        return pca_result
```

```

def LDA(data,target, n):
        from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
        lda = LDA(n_components=n)
        lda_result = lda.fit_transform(data,target)
        return lda_result

def plot(data,target, n):
        pca_result = PCA(data, n)
        lda_result = LDA(data,target, n)

        plt.figure(figsize=(16,9))
        plt.subplot(1, 2, 1)
        for color,target_name in zip("rgb", ['Iris setosa','Iris versicolor','Iris
            virginica']):
                plt.scatter(np.array(pca_result[iris_target == target_name, 0]), np.
                    array(pca_result[iris_target == target_name, 1]), c=color, label=
                    target_name)
        plt.title('PCA on iris')

        plt.subplot(1, 2, 2)
        for color,target_name in zip("rgb",      ['Iris setosa','Iris versicolor','Iris
            virginica']):
                plt.scatter(np.array(lda_result[iris_target == target_name, 0]), np.
                    array(lda_result[iris_target == target_name, 1]), c=color, label=
                    target_name)
        plt.title('LDA on iris')
        plt.show()

plot(X,iris_target,2)
```

Result:

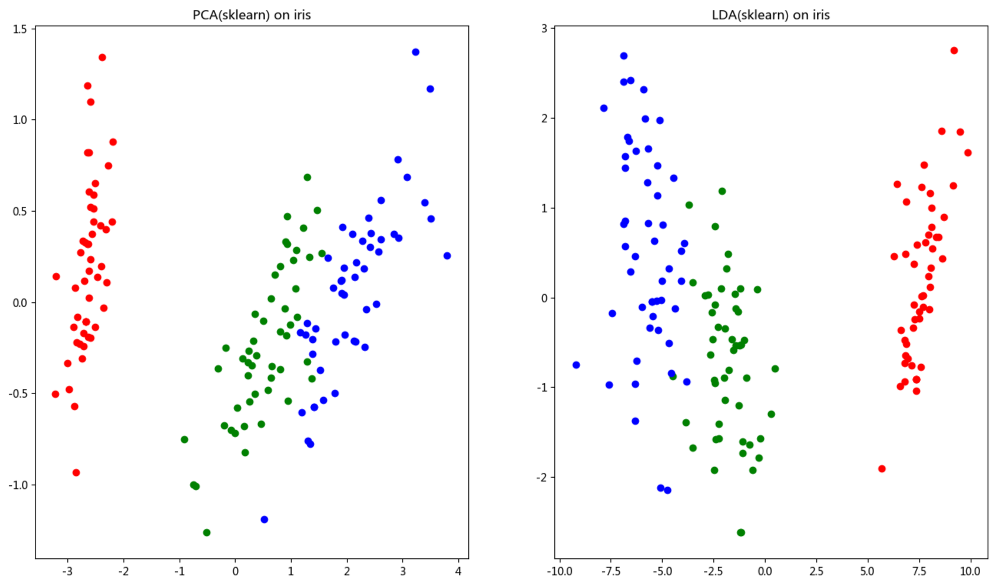


Figure 2.10: PCA(sklearn) and LDA(sklearn)

Comparing Figure 2.9 and Figure 2.10, we can see that the results of the two are roughly the same, but they are symmetrical to each other. This is because the PCA and LDA in the class learn are realized through the svd_flip function. Sklearn performs a process on the singular value decomposition results. The results obtained by inverting u and v at the same time are consistent. The final result is correct, but the specific value There are some differences due to differences in algorithms.

Looking at Figure 2.9, we can see that both PCA and LDA reduce the original 4D data to 2D, but in PCA, the boundaries of the two categories of Iris-versicolor and Iris-virginica overlap more, compared to the effect of LDA Better. This is because LDA is a supervised learning, which will consider the label of the data, so as to keep the data of different labels as far away as possible, while PCA for unsupervised learning will only consider the data weight ratio.