EEE418: Advanced Pattern Recognition **Spring 2020**

Lab 1: Feature Extraction

Student: Xun Xu *ID: 1926930*

Chapter 1

Introduction: Theory Introduction

In many cases, there may be correlations between many variables, thereby increasing the complexity of problem analysis. If each indicator is analyzed separately, the analysis is often isolated and the information in the data cannot be fully utilized. Therefore, blindly reducing the indicators will lose a lot of useful information, resulting in erroneous conclusions. Therefore, it is necessary to find a reasonable method to reduce the loss of the information contained in the original indicators while reducing the indicators that need to be analyzed, so as to achieve the purpose of comprehensive analysis of the collected data. Since there is a certain correlation between the variables, you can consider changing closely related variables into as few new variables as possible, so that these new variables are uncorrelated, then you can use less comprehensive indicators to represent each Various types of information existing in various variables. Principal component analysis and lda are two typical data dimensionality reduction methods.

1.1 PCA and principle derivation

As shown in the figure1.1, we take two-dimensional data as an example. If we can find a reasonable direction u as the projection direction of the data, the error between all the original points and the projection point is minimized, so we use to store the data parameters It can be reduced from 2n to n+2, the next thing we need to do is to reduce the minimum reconstruction error, as AA'and BB' shown in Figure1.1.

The minimum reconstruction error is shown as $\vec{e}$, which is the vector difference between the original point and the projected point. According to the derivation of the formula, we can know that the smallest reconstruction error is to maxi-

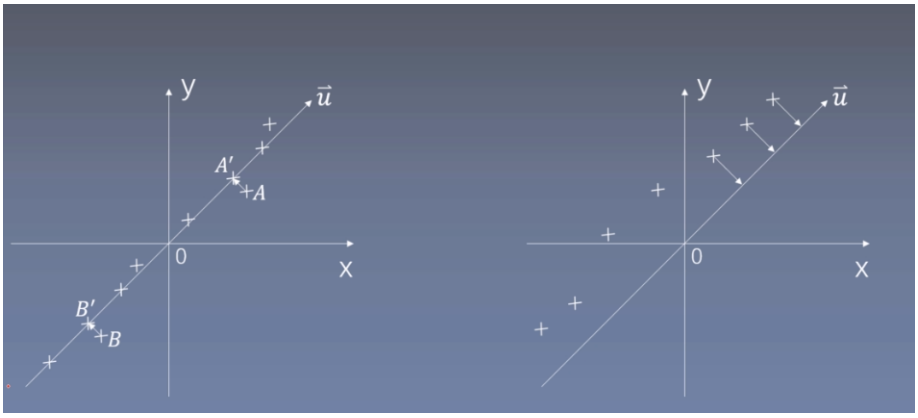

Figure 1.1: Data projection compression of PCA

mize $\left(x^T u\right)$,where $\vec{e}$ is the difference between the original vector and the projected vector, p_{rj} is the projected length, and $\vec{u}$ is the projected direction

$$\begin{aligned}
\vec{e} &= \vec{x} - p_{rj}\vec{x} \\
&= \vec{x} - \langle \vec{x}, \vec{u} \rangle \vec{u} \\
&= x - \left(x^T u\right) u; x, u \in \mathbb{R}^n \quad and \|u\| = 1, u^T u = 1 \\
&= \|\vec{e}\|^2 = e^T e = \left[x - \left(x^T u\right) u\right]^T \left[x - \left(x^T u\right) u\right] \\
&= \left[x^T - \left(x^T u\right) u^T\right] \left[x - \left(x^T u\right) u\right] \\
&= x^T x - \left(x^T u\right)\left(x^T u\right) - \left(x^T u\right)\left(u^T x\right) + \left(x^T u\right)^2 u^T u \\
&= \|x\|^2 - \left(x^T u\right)^2 - \left(x^T u\right)^2 + \left(x^T u\right)^2 \\
&= \|x\|^2 - \left(x^T u\right)^2
\end{aligned}$$

After expanding $\left(x^T u\right)$, we can find that the maximum value is actually the maximum value$\left(x_i^T x_i\right)$.

$$\max\left(x^T u\right)^2 \Leftrightarrow \max\left(x^T u\right)\left(x^T u\right) \Leftrightarrow \max\left(u^T x\right)\left(x^T u\right) \Leftrightarrow \max u^T \left(x x^T\right) u$$

$$\max \sum_{i=1}^{N} u^T \left(x_i x_i^T\right) u = u^T \left(\sum_{i=1}^{N} x_i x_i^T\right) u, and \|u\| = 1$$

$$\max u^T (X) u, st : \|u\| = 1$$

From the following formula, we can know that we can use the Lagrange multiplier method to find the extreme value, and then the partial derivative from the matrix can get the final $\left(x^T x\right)$ feature vector direction we require is the $\vec{u}$ direction.

$$L(u, \lambda) = u^T X u + \lambda\left(1 - u^T u\right)$$
$$\frac{\partial L}{\partial u} = 0 \Rightarrow Xu - \lambda u = 0$$
$$Xu = \lambda u$$
$$\frac{\partial L}{\partial u} = 0 \Rightarrow u^T u = 1$$

Finally, we can derive that the PCA method is essentially to find the eigenvalue and eigenvector of $\left(x^T x\right)$.

1.2 LDA and principle derivation

The basic idea of LDA is relatively simple. As shown in the figure1.2, given a labeled training sample set, try to project the sample onto a straight line, so that the projection points of similar samples are as close as possible, and the projection points of heterogeneous samples are as far as possible.That is, within-class distance is small, between-class distance is large.

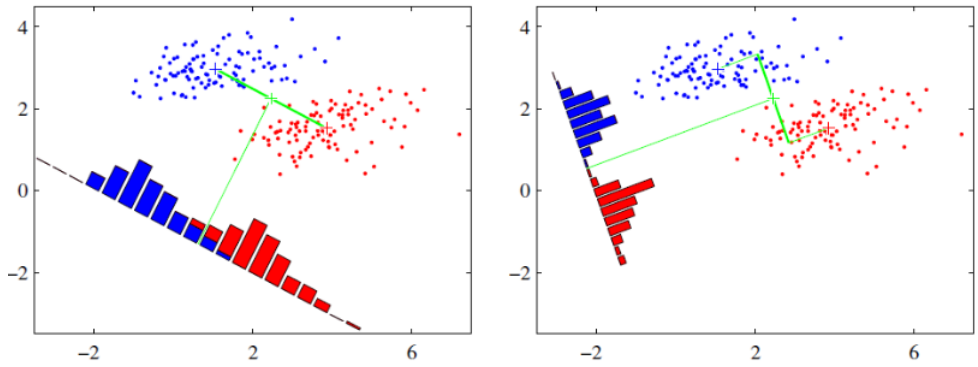

Figure 1.2: LDA principle introduction

Then given a K-type sample, the number of samples in each type is: $N_1 ... N_k$, what we need to do is to minimize intra-class errors and maximize inter-class errors, The in-class error of the k-th sample can be expressed as:

$$\begin{aligned}
\mathrm{S_k} &= \sum_{x \in D_k} (\bar{x} - \widetilde{\mathrm{m}})^T (\bar{x} - \widetilde{\mathrm{m}}) \\
&= \sum_{x \in D_k} \left[\left(x^T u\right) u^T - \left(m^T u\right) u^T \right] \left[\left(x^T u\right) u - \left(m^T u\right) u \right] \\
&= \sum_{x \in D_k} \left[\left(x^T u\right)^2 u^T u - 2 \left(x^T u\right) \left(m^T u\right) u^T u + \left(m^T u\right)^2 u^T u \right] \\
&= \sum_{x \in D_k} \left[\left(x^T u\right)^2 - 2 \left(x^T u\right) \left(m^T u\right) + \left(m^T u\right)^2 \right]
\end{aligned}$$

where m is the sample mean of each class, u is the projection direction, x is the original sample and $\bar{x}$is the projected sample.

We further process the formula and find the mean of S_k:

$$\mathrm{S_k} = \sum_{x \in D_t} \left[\left(x^T u\right)^2 - 2\left(x^T u\right)\left(m^T u\right) + \left(m^T u\right)^2 \right]$$

$$\begin{aligned}
\frac{\mathrm{S_k}}{N_k} =& \frac{\sum_{x \in D_t} \left(x^T u\right)^2}{N_k} - 2\frac{\sum_{x \in D_t} x^T \left(u m^T u\right)}{N_k} + \frac{\sum_{x \in D_t} \left(m^T u\right)^2}{N_k} \\
&= \frac{\sum u^T x x^T u}{N_k} - 2\frac{\sum x^T}{N_k} u m^T u + \left(m^T u\right)^2 \\
&= u^T \frac{\sum x x^T}{N_k} u - \left(m^T u\right)^2 \\
&= u^T \frac{\sum x x^T}{N_k} u - u^T m m^T u \\
&= u^T \left(\frac{\sum x x^T}{N_k} - m m^T \right) u
\end{aligned}$$

From this, the intra-class error function can be obtained,as follows:

$$\begin{aligned}
\sum_k S_k &= \sum_k u^T \left(\frac{\sum x x^T}{N_k} - m_k m_k^T \right) u \\
&= u^T \prod_L^T \frac{\sum x x^T}{N_k} - m_k m_k^T | u \\
&= u^T S_w u
\end{aligned}$$

For the intra-class error, it is relatively simple and its derivation is as follows:

$S_{ij} = \left(\widetilde{m}_i - \widetilde{m}_j\right)^T \left(\widetilde{m}_i - \widetilde{m}_j\right)$

$= u^T \left(m_i - m_j\right)^T \left(m_i - m_j\right) u$

$\sum_{i,j} S_{ij} = u^T \prod_{i,j} \left(m_i - m_j\right)^T \left(m_i - m_j\right)$

$= u^T S_b u$

$(i \neq j)$

From the formula and the derivation of the formula, we know that to minimize the intra-class error is to make s_w minimum, At the same time, the inter-class error is the direct difference of the sample mean of different classes. Similarly, the condition that maximizes the inter-class error can be obtained, that is, the s_b is maximized. So we can get what we want, that is, to maximize J(u), which is known by the Lagrange multiplier method, that is, to find the eigenvalue and eigenvector of $S_b S_w^{-1}$.

$J(u) = \frac{u^T S_b u}{u^T S_w u}$

$L(u, \lambda) = u^T S_b u + \lambda \left(1 - u^T S_w u\right)$ $\frac{\partial L}{\partial u} = 0 \rightarrow S_b u = \lambda S_w u$ $S_b S_w^{-1} u = \lambda u$

Chapter 2

Methodology

Since the amount of code for this assignment is not large, I present the content required by the question in code form, and important parts are commented.

1.1 PCA code

```
#
import numpy as np
import matplotlib.pyplot as plt

# Read file
f = open('./iris.data', 'r+')
dataset = f.readlines()
f.close()

# Convert string data to numeric data and create data sets and
                                    label
data = np.zeros((len(dataset)-1,4))
label = []
for  i in range(150):
k = dataset[i][16:-1]
label.append(k)
m = float(dataset[i][0:3])
n= float(dataset[i][4:7])
p = float(dataset[i][8:11])
q = float(dataset[i][12:15])
data[i] = np.asarray([m, n, p, q])
dataset = data

# PCA
X = data
k = 2 #Compressed to k-dimensional data
n_samples = X.shape[0]
```

```
n_features = X.shape[1]
mean = np.array([np.mean(X[:, i]) for i in range(n_features)])

# normalization:Standardize the data
norm_X = X - mean
print('norm_X',norm_X)
# Compute the covariance matrix
covariance_matrix = np.dot(np.transpose(norm_X), norm_X)
print('covariance_matrix')
print(covariance_matrix.shape)
# Calculate the eigenvectors and eigenvalues
eig_val, eig_vec = np.linalg.eig(covariance_matrix)
print( eig_vec, eig_val)
eig_pairs = [(np.abs(eig_val[i]), eig_vec[:, i]) for i in range(
                                    n_features)]
# sort eig_vec based on eig_val from highest to lowest
eig_pairs.sort(reverse=True)
# select the top k eig_vec
feature = np.array([ele[1] for ele in eig_pairs[:k]])

# get new data
data = np.dot(norm_X, np.transpose(feature))
print('data_pca',data)

# pca drawing data
y1 = data

# plot the data
# Categorize data based on tags
data1 = np.array([y1[i] for i in range(len(y1)) if label[i] == '
                                    Iris-setosa'])
data2 = np.array([y1[i] for i in range(len(y1)) if label[i] == '
                                    Iris-versicolor'])
data3 = np.array([y1[i] for i in range(len(y1)) if label[i] == '
                                    Iris-virginica'])
# Drawing part
plt.scatter(data1[:,0], data1[:,1], alpha=0.6,marker='x', label='
                                    Iris-setosa')
plt.scatter(data2[:,0], data2[:,1], alpha=0.6,marker='.',label='
                                    Iris-versicolor')
plt.scatter(data3[:,0], data3[:,1], alpha=0.6,marker='*',label='
                                    Iris-virginica')
plt.legend()
plt.title('PCA')
plt.show()
```

1.2 LDA code

```
#
#LDA
data_x = dataset
data_y = label
n_sample = data_x.shape[0]
n_feature = data_x.shape[1]
# Generate {Xk} matrix according to the labels of the examples
data1 = np.array([data_x[i] for i in range(len(data_x)) if
                                data_y[i] == 'Iris-setosa'])
data2 = np.array([data_x[i] for i in range(len(data_x)) if
                                data_y[i] == 'Iris-versicolor'])
data3 = np.array([data_x[i] for i in range(len(data_x)) if
                                data_y[i] == 'Iris-virginica'])
# Compute the class-wise mean and the total mean of the data
                                matrix
u = np.mean(data_x, axis=0, keepdims=True)
u1 = np.mean(data1, axis=0, keepdims=True)
u2 = np.mean(data2, axis=0, keepdims=True)
u3 = np.mean(data3, axis=0, keepdims=True)
# Compute the within-class covariance matrix SW
sw = np.dot(np.transpose(data1 - u1), data1 - u1) + np.dot(np.
                                transpose(data2 - u2), data2 - u2
                                ) + np.dot(np.transpose(data3 -
                                u3), data3 - u3)
print('sw')
print(sw)
# Compute the between-class covariance matrix SB
sb = data1.shape[0] * np.dot(np.transpose(u1 - u), u1 - u) +
                                data2.shape[0] * np.dot(np.
                                transpose(u2 - u), u2 - u) +
                                data3.shape[0] * np.dot(np.
                                transpose(u3 - u), u3 - u)
print('sb')
print(sb)
# sw^-1 * sb
C = np.linalg.inv(sw) * sb
# D = P * C * P^T
eig_val, eig_vec = np.linalg.eig(C)
print(eig_val, eig_vec)
# Find projection eigvector using method 2
sorted_idx = np.argsort(eig_val)[::-1]
eig_val_new = eig_val[sorted_idx]
eig_vec_new = eig_vec[:, sorted_idx]
y = np.dot(data_x,np.transpose(eig_vec))
# lda drawing data
y2 = y[:,0:2]
```

```
# plot the data
# Categorize data based on tags
data1 = np.array([y2[i] for i in range(len(y2)) if data_y[i] ==
                                'Iris-setosa'])
data2 = np.array([y2[i] for i in range(len(y2)) if data_y[i] ==
                                'Iris-versicolor'])
data3 = np.array([y2[i] for i in range(len(y2)) if data_y[i] ==
                                'Iris-virginica'])
# Drawing part
plt.scatter(data1[:,0], data1[:,1], alpha=0.6,marker='x', label=
                                'Iris-setosa')
plt.scatter(data2[:,0], data2[:,1], alpha=0.6,marker='.',label='
                                Iris-versicolor')
plt.scatter(data3[:,0], data3[:,1], alpha=0.6,marker='*',label='
                                Iris-virginica')
plt.title('LDA')
plt.legend()
plt.show()
```

Chapter 3

Analysis of results and comparison

According to the two figiures figure1.11 and figure1.12:
1 Both can reduce the dimension of the data.
2 Both of them use the matrix feature decomposition when reducing dimensions.

But these two methods may have a big difference when faced with different types of data, they have the following differences:
1 Because LDA is a supervised dimensionality reduction method, and PCA is an unsupervised dimensionality reduction method, PCA did not use the original category information of the data at the beginning, resulting in different categories of data overlapping after dimensionality reduction and losing some Useful information, and LDA does not have such problems
2 LDA can be used for classification in addition to dimensionality reduction
3 However, PCA is usually much smaller than lda in calculation, which can be reflected in our algorithm, so pca is more widely used in data dimensionality reduction.
4 In principle, we know that LDA selects the projection direction with the best classification performance, and PCA selects the direction of the sample point projection with the largest variance. Lda has better performance for data with linear relationship, but when the data distribution is uneven (with Nonlinear relationship), lda has better performance.

For our data set, the difference between the two can reflect the above differences, but it is not particularly obvious.

```
norm_X
 [[-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]
 [-4.43333333e-01  8.46000000e-01 -2.45866667e+00 -7.98666667e-01]
 [-7.43333333e-01  4.46000000e-01 -2.35866667e+00 -8.98666667e-01]
 [-1.43333333e-01  7.46000000e-01 -2.05866667e+00 -8.98666667e-01]
```

Figure 1.1: data after standardized(PCA)

```
covariance_matrix
[[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 1.2: covariance matrix of PCA

```
new eig_vec
 [[ 0.36158968 -0.08226889  0.85657211  0.35884393]
 [-0.65653988 -0.72971237  0.1757674   0.07470647]]
```

Figure 1.3: Reordered eigenvector of PCA

```
new data
 [[-2.68420713 -0.32660731]
 [-2.71539062  0.16955685]
 [-2.88981954  0.13734561]
 [-2.7464372   0.31112432]
 [-2.72859298 -0.33392456]
 [-2.27989736 -0.74778271]
 [-2.82089068  0.08210451]
 [-2.62648199 -0.17040535]
 [-2.88795857  0.57079803]
 [-2.67384469  0.1066917 ]
 [-2.50652679 -0.65193501]
 [-2.61314272 -0.02152063]
```

Figure 1.4: training dataset after dimensionality reduction(PCA)

```
Generate Xk matrix
(50, 4) (50, 4) (50, 4)
```

Figure 1.5: generate data of different types(PCA)

```
the first type class-wise mean
2.533
the second type class-wise mean
3.573
the third type class-wise mean
4.284999999999999
total mean
3.4636666666666684
```

Figure 1.6: class-wise mean and total mean of the data matrix(LDA)

```
sw
[[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 1.7: the within-class covariance matrix S_b

```
sb
[[ 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 1.8: the between-class covariance matrix S_w

```
eig_vec_new
[[ 0.21393983  0.1482758  -0.90254529 -0.34301026]
 [ 0.03251495 -0.05736353 -0.35598315  0.93216318]
 [-0.91045659 -0.32775137 -0.2393291  -0.07980862]
 [ 0.35247883 -0.9312909  -0.03754438 -0.08394252]]
```

Figure 1.9: Reordered eigenvector of LDA

```
data_LDA
 [[ 2.77892984e-01 -3.46889902e-01]
 [ 1.60967117e-01 -3.24711126e-01]
 [ 2.38088841e-01 -3.07088506e-01]
 [ 2.13582195e-02 -3.75800278e-01]
 [ 2.71326582e-01 -3.55877749e-01]
 [ 6.20196162e-02 -2.80443138e-01]
 [ 1.21794464e-01 -2.64194703e-01]
 [ 1.51416892e-01 -3.80003358e-01]
 [ 3.91696219e-02 -3.35232246e-01]
 [ 1.19841194e-01 -4.59262112e-01]
 [ 2.81475565e-01 -3.84206439e-01]
 [ 1.83743974e-02 -4.22104662e-01]
 [ 1.73874160e-01 -4.21178939e-01]
 [ 3.37667832e-01 -3.30641467e-01]
```

Figure 1.10: training dataset after dimensionality reduction(LDA)

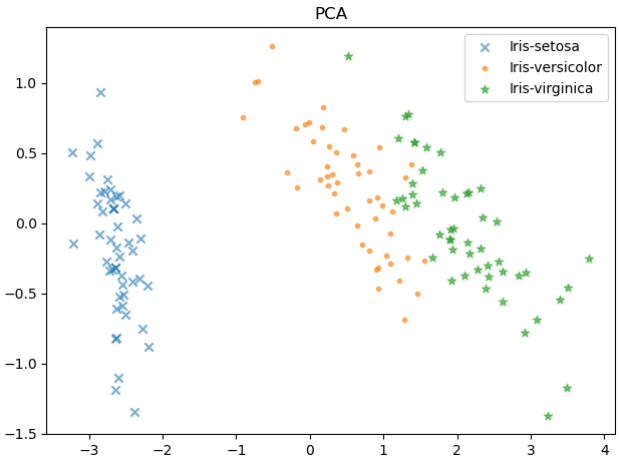

Figure 1.11: new dataset of PCA

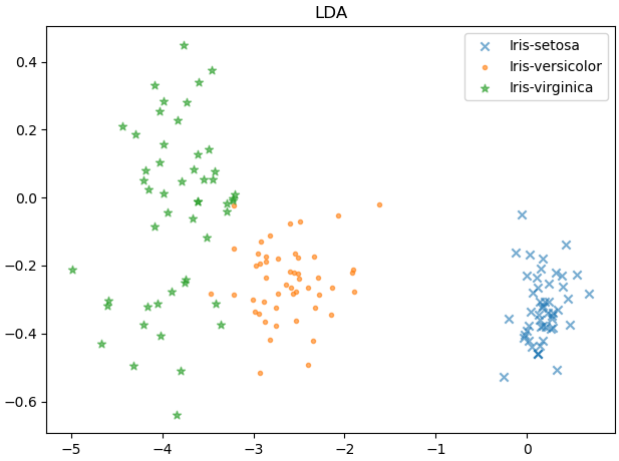

Figure 1.12: new dataset of LDA