Project1-Part 2

天津大學

The LDA and KLDA transformations of data

电　　话 13312055621

专　　业 水利水电工程

课程名称 机器学习算法与应用

班　　级 9 班

学　　号 2018205075

姓　　名 董志明

2018 年 11 月 4 日

Abstract

It should be an explicit summary of the report that states the problem, the methods used, the major experiments, discussions and conclusions.

1. Introduction

LDA(Linear Discriminant Analysis)，中文名为线性判别分析，是一种经典的降维方法，LDA 在模式识别领域（比如人脸识别，舰艇识别等图形图像识别领域）中有非常广泛的应用，因此我们有必要了解下它的算法原理。

LDA 是一种监督学习的降维技术，也就是说它的数据集的每个样本是有类别输出的。这点和 PCA 不同。PCA 是不考虑样本类别输出的无监督降维技术。LDA 的思想可以用一句话概括，就是“投影后类内方差最小，类间方差最大”。什么意思呢？ 我们要将数据在低维度上进行投影，投影后希望每一种类别数据的投影点尽可能的接近，而不同类别的数据的类别中心之间的距离尽可能的大。

2. LDA and KLDA

2.1 Theory of LDA and KLDA

In this section, the background, principle and derivation formula on LDA and KLDA should be described soundly.

LDA 的原理是找到一个满足如下特征的向量：

类间均值在该向量上的投影最大，每一个类中的样本投影与相应类均值投影之间的距离最小。

第 i 类的均值是：

$$\overline{m_i} = \frac{1}{N_i}\sum_{j=1}^{N_i} v^T x_j^{(i)} = v^T\left(\frac{1}{N_i}\sum_{j=1}^{N_i} x_j^{(i)}\right) = v^T m_i$$

类内的平方和距离可以表示为：

$$\sum_{i=1}^{L-1}\sum_{j=i+1}^{L}\frac{N_i}{N}\frac{N_j}{N}(\overline{m_i}-\overline{m_j})^2 = \sum_{i=1}^{L-1}\sum_{j=i+1}^{L}\frac{N_i}{N}\frac{N_j}{N}(\overline{m_i}-\overline{m_j})(\overline{m_i}-\overline{m_j})^T = v^T S_b^{LDA} v$$

整理得：

$$S_b^{LDA} = \sum_{i=1}^{L}\frac{N_i}{N}(m_i - m_0)(m_i - m_0)^T$$

所有类中所有样本的方差为：

$$\sum_{i=1}^{L}\sum_{j=1}^{N_i}\frac{1}{N}(v^T x_j^{(i)} - \overline{m_i})^2 = \sum_{i=1}^{L}\sum_{j=1}^{N_i}\frac{1}{N}(v^T x_j^{(i)} - \overline{m_i})(v^T x_j^{(i)} - \overline{m_i})^T = v^T S_w^{LDA} v$$

整理得：

$$S_w^{LDA} = \sum_{i=1}^{L}\sum_{j=1}^{N_i}\frac{1}{N}(x_j^{(i)} - m_i)(x_j^{(i)} - m_i)^T$$

因此，要求的向量可以通过如下等式求得：

$$v = \underset{v^T S_w^{LDA} v=1}{argmax} v^T S_w^{LDA} v$$

通过拉格朗日乘子法即可求解。

基于 kernel 的 LDA:

将原式中的 x 替换为φ(x)，经过相同的推导过程，可以得到：

$$S_b^{GDA} = \frac{1}{N} X^T B X$$

$$S_w^{GDA} = \frac{1}{N} X^T X$$

式中，$B = \begin{bmatrix} B_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & B_L \end{bmatrix}$，$X^T = [X_1^T, \ldots\ldots, X_L^T]$

这等于求解如下的特征值问题：

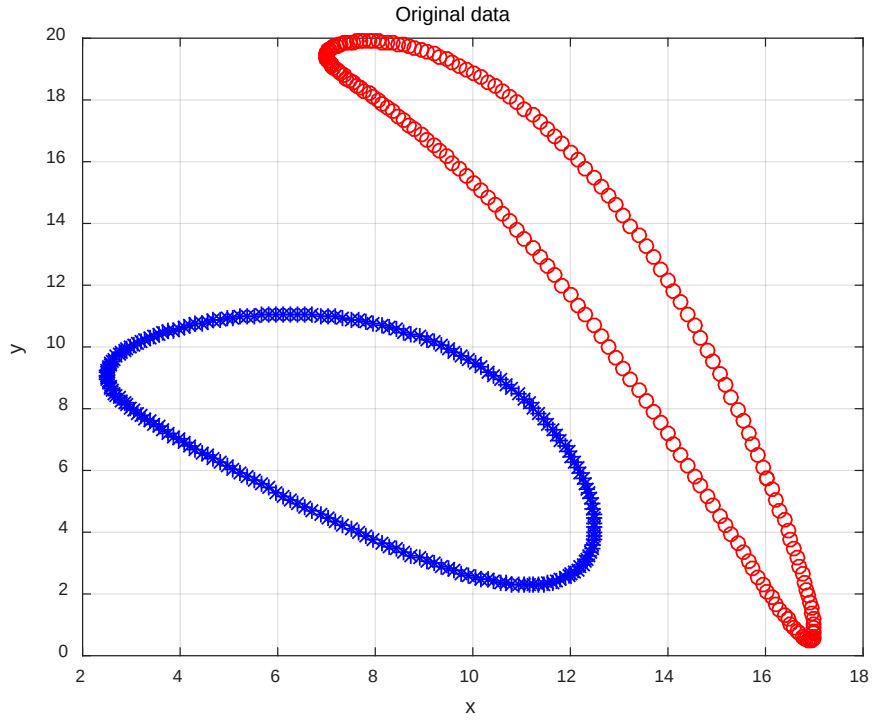
$$\left(\frac{1}{N} X^T B X\right) v = \lambda (S_b^{GDA} = \frac{1}{N} X^T X) v$$

最后，数据在新的投影空间内可以表示为：

$$v^T \phi(x) = a^T \begin{bmatrix} K(x_1, x) \\ \vdots \\ K(x_N, x) \end{bmatrix}$$

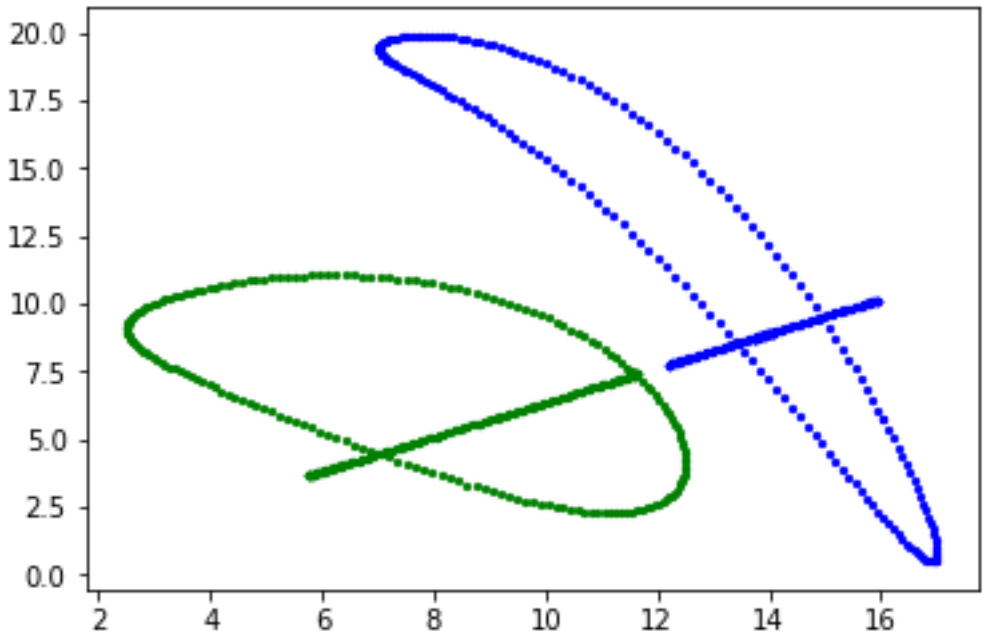
2.2 The LDA and KLDA for "data_LDA.txt"

In this section, an actual project about classification of data sets is required to carry out with LDA and KLDA. The steps and results of processing should be given in detail. The original data was given as follows, the data from different classes in which were marked as red (1st-201th data in "data_LDA.txt") and blue (202th-402th data in "data_LDA.txt") respectively.

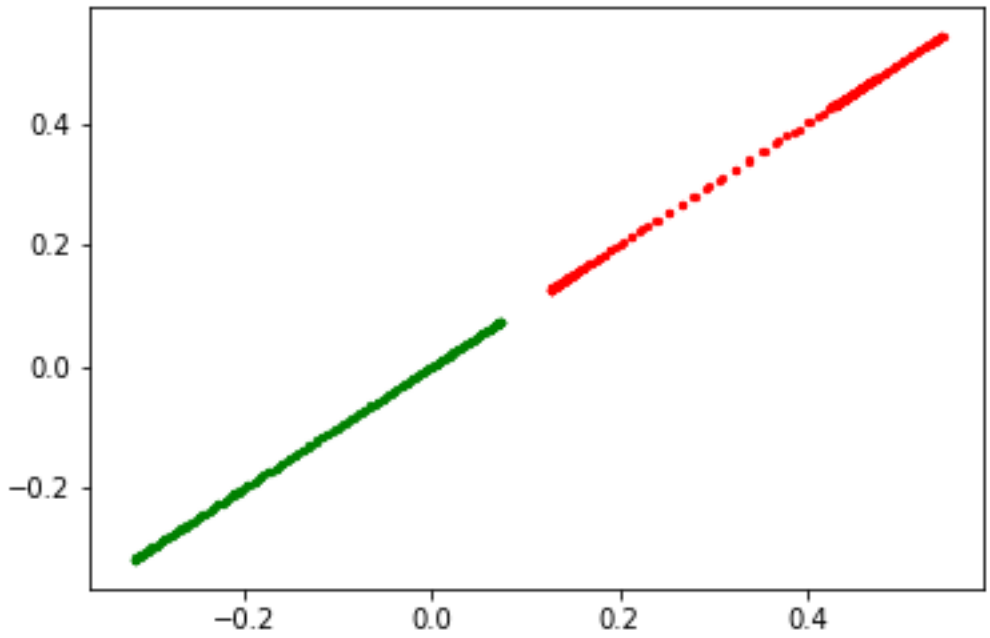


3. Conclution

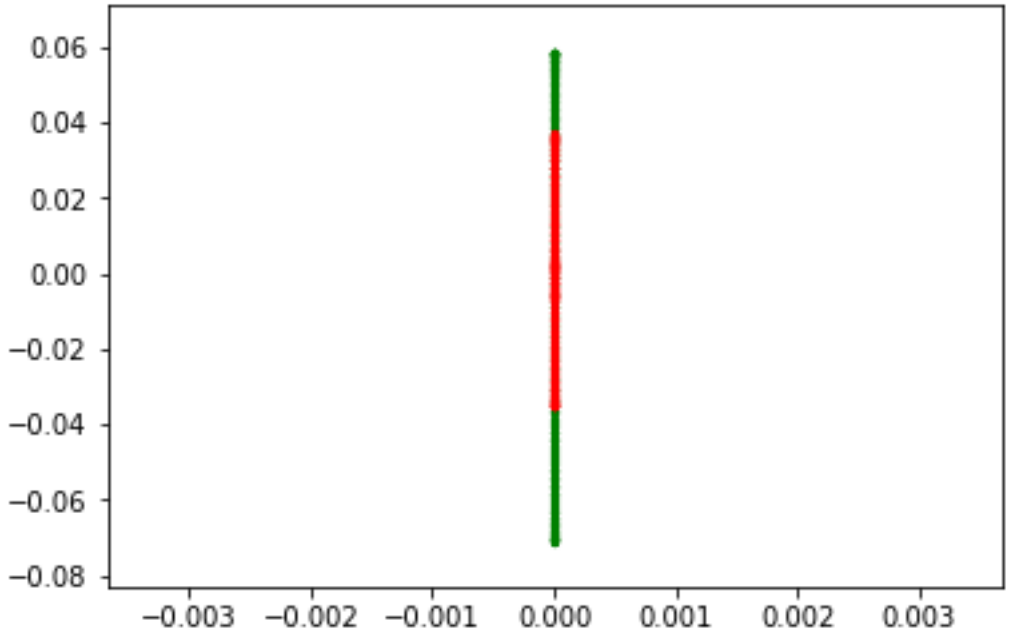
LDA 降维结果（一维），用 python 实现。由图可知，降维的结果非常直观，数据也被完美的分成了两类。



kLDA 降维结果（二维），用 python 实现，如图为基于径向基核函数的降维结果，取前两个特征向量，当 para 的参数取为 7 的时候，取得了比较好的结果，如下图所示：



但是，当para变化时，微小的变化就会出现完全不同的结果，para的选取具有偶然性，比如，当para=7.0001时，降维的结果就变成了下图：



猜想的原因可能是因为原来的K矩阵很小，但经过变换并且求逆之后就变得特别大，所以微小的变动都会导致巨大的误差，因此尝试进行归一化处理，效果依然不好。

Appendix

Give the used Codes

Specify your environment and how to execute.

Lda：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 26 18:45:52 2018

@author: 11854
"""
import numpy as np
import matplotlib.pyplot as plt
#读取数据
filename='data_LDA.txt'
train_data = np.loadtxt(filename, delimiter=',', dtype=np.str)
train_data = np.float64(train_data)

data_num=len(train_data)
class_num=2
each_class_num=data_num/class_num
each_class_num=int(each_class_num)
m=np.zeros([2,2])
m0=np.mean(train_data,axis=0)
```

```
S_LDA_w=np.zeros([2,2])
#求各个类的均值,求类内方差矩阵S_LDA_w
for i in range(class_num):
    temp_data=train_data[i*each_class_num:(i+1)*each_class_num,:]
    m[i]=np.mean(temp_data,axis=0)
    S_LDA_w=S_LDA_w+np.dot((temp_data-m[i]).T,(temp_data-m[i]))/data_num

    #m的每一行是一个类的均值


#求类间的方差S_LDA_b,由于每类样本数一样，所以这里可以简化计算
S_LDA_b=np.zeros([2,2])
S_LDA_b=S_LDA_b+np.dot((m-m0).T,(m-m0))/2

#求特征值，特征向量
eigenvalue,featurevector=np.linalg.eig(np.linalg.inv(S_LDA_w).dot(S_LDA_b))

#求最大特征值对应的特征向量
newspace=featurevector[:,np.argmax(eigenvalue)]

##new_data_1=(train_data[0:201,:]).dot(newspace)
#new_data_2=(train_data[201:402,:]).dot(newspace)

k=newspace[1]/newspace[0]
new_data=train_data.dot(newspace)
new_data_map=np.zeros_like(train_data)
for i in range(data_num):
    new_data_map[i][0]=train_data[i].dot([[1],[k]])/(k*k+1)
    new_data_map[i][1]=new_data_map[i][0]*k

#画图
plt.scatter(train_data[0:201,0], train_data[0:201,1], s=5,c='g')
plt.scatter(train_data[201:402,0], train_data[201:402,1], s=5,c='b')

plt.scatter(new_data_map[0:201,0], new_data_map[0:201,1], s=5,c='g')
plt.scatter(new_data_map[201:402,0], new_data_map[201:402,1], s=5,c='b')
plt.show()
```

klda

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 29 00:59:26 2018
```

```
@author: 11854
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
import math
dim=2

#读取数据
filename='data_LDA.txt'
train_data = np.loadtxt(filename, delimiter=',', dtype=np.str)
train_data = np.float64(train_data)

#train_data=preprocessing.scale(train_data)
#K=kernel(np.vstack((train_data1_mean,train_data2_mean)))
data_mean=train_data
#data_mean=train_data
nun_of_data=data_mean.shape[0]
distance_maxtrix=np.zeros([nun_of_data,nun_of_data])
K=np.zeros([nun_of_data,nun_of_data])
K0=np.zeros([nun_of_data,nun_of_data])
#k是kernel矩阵
a=np.sum(np.square(data_mean),axis=1,keepdims=True)
#生成一个一行500列的向量，每一个元素是每张图片3072个像素点数据的平方和,x是测试集,y^2
b=np.sum(np.square(data_mean),axis=1)
#生成一个一行5000行的向量，每一个元素是每张图片3072个像素点数据的平方和,x是训练集x^2
c=np.multiply(-2,np.dot(data_mean,data_mean.T))
#distance_maxtrix=np.add(a,b)
#distance_maxtrix=np.add(distance_maxtrix,c)
#distance_maxtrix=np.sqrt(distance_maxtrix)



kernel_sigma=7.0001


#算距离矩阵
for i in range(nun_of_data):
    for j in range(nun_of_data):
        distance_maxtrix[i][j]=np.linalg.norm(data_mean[i]-data_mean[j])
        K0[i][j]=math.exp(-distance_maxtrix[i][j]**2/2/kernel_sigma**2)
```

```
oneN=np.ones_like(K0)/nun_of_data
#oneN=np.ones_like(K0)
K=K0-oneN.dot(K0)-K0.dot(oneN)+(oneN.dot(K0)).dot(oneN)

B=np.zeros_like(K)
B[0:201,0:201]=np.ones((201,201))/201
B[201:402,201:402]=np.ones((201,201))/201

#提取特征值，特征向量
ddd=np.linalg.inv(K0.dot(K0))
ccc=(K0.dot(B)).dot(K0)
#ccc=((np.linalg.inv(K0)).dot(B)).dot(K0)
eigenvalue,featurevector=np.linalg.eig(ddd.dot(ccc))
featurevector=featurevector.real
eigenvalue=eigenvalue.real


eigenvalue_index=np.argsort(-eigenvalue)
eigenvalue_index=eigenvalue_index[0:dim]
#按照降序排列，返回索引
#eigenvalue_sort1=eigenvalue[np.argsort(-eigenvalue)]
eigenvalue_sort=eigenvalue[eigenvalue_index[0:dim]]
featurevector_sort=featurevector[:,eigenvalue_index[0:dim]]

data_mapped=(featurevector_sort.T).dot(K0)
data_mapped=data_mapped.T
plt.scatter(data_mapped[0:201,0], data_mapped[0:201,1], s=5,c='g')
plt.scatter(data_mapped[201:402,0], data_mapped[201:402,1], s=5,c='r')
#plt.scatter(data_mapped[0:201,0], data_mapped[0:201,0], s=5,c='g')
#plt.scatter(data_mapped[201:402,0], data_mapped[201:402,0], s=5,c='r')
plt.show()
```