目录

[一、数据、问题描述 2](#_Toc518305203)

[二、选用方法的理由 3](#_Toc518305204)

[三、特征提取 4](#_Toc518305205)

[四、分类器设计 6](#_Toc518305206)

[五、评价方法 7](#_Toc518305207)

[六、具体实现 10](#_Toc518305208)

[七、测试结果 11](#_Toc518305209)

[八、结论 17](#_Toc518305210)

[参考文献 18](#_Toc518305211)

[附录 19](#_Toc518305212)

# 一、数据、问题描述

在本次课程设计中，我所使用的数据是来自于UCI数据集的手写数字光学识别数据集[1]。一共有43个人提供了他们的手写数字（数字范围为0到9），其中30个人的手写数字被作为训练集数据，另外13个人的手写数字被作为测试集数据。数据提供者将数字写在格式纸上。预处理过程中，测试机构将每一个手写数字以大小为32×32位图的形式提取出来，并对每个像素进行了归一化处理；每张位图被分成64个大小为4×4的不重叠位图块，测试机构统计了每个位图块中被字迹覆盖的像素的个数，这样每张手写数字位图便产生了64个特征，每个特征数据都为整数，范围在1至16之间。训练集与测试集都是每行行数据为一个样本，一行数据有65列，前64列为特征数据，第65列为样本标签（即这个手写数字是几）；训练集中共有3823个样本，其中标签为0到9的手写数字样本个数分别为376，389，380，389，387，376，377，387，380，382；测试集共有1797条数据，其中标签为0到9的手写数字样本个数分别为178，182，177，183，181，182，181，179，174，180。可以看出样本分布是较为均衡的。

# 二、选用方法的理由

在模式识别中，Fisher线性判别分析 (Fisher's linear discriminant) 是最直观且具有代表性的线性判别方法之一。在面对c个类的判别时，其意图是将d维空间中的数据点投影到c-1维空间上去，使得不同类的样本点在这个空间上的投影尽量分离，同类的尽量紧凑。在特征维数d较大而类别数c较小的时候，Fisher线性判别分析可以有效降低特征维度，减少计算量。同时，与主成分分析 (Principal Components Analysis,PCA) 相比，Fisher线性判别分析可以利用已知样本的类别信息，是一种有监督的降维方法。[2]

因为数据集当中各类别的分布较为均衡，且我是初次从底层实现一个模式识别算法，故而选择了这一种较为简单、直观的算法来实现。

# 三、特征提取

LDA (linear discriminant analysis)[3] 是一种有监督的对特征进行降维的方法。

在面对c个类的判别时，所要做的是将d维空间中的数据点向c-1维空间投影。从d维空间向c-1维空间投影是通过c-1个投影方程进行的：

( 3.1 )

这里的为第类的样本集。设，c-1个方程可以更简练地表达：

( 3.2 )

这里的为第i类的样本的投影向量集。类间离散度矩阵和类内离散度矩阵可以由总体离散度矩阵和总体均值向量m推导得到：

( 3.3 )

( 3.4 )

( 3.5 )

由此定义类间离散度矩阵和类内离散度矩阵：

( 3.6 )

( 3.7 )

( 3.8 )

那么样本数据的投影向量的类间离散度矩阵和类内离散度矩阵即为：

( 3.9 )

( 3.10 )

要找到某一W使得类内离散度尽量小，类间离散度尽量大。这里的类内离散度和类间离散度不是一个值，而是一个矩阵。矩阵的行列式是矩阵的特征值的乘积，也就是数据在各个主要方向的方差的积，相当于类别离散度超椭球体的体积的平方。故使用行列式来度量离散度，这样判别函数为：

( 3.11 )

可以证明，当W的列向量是的广义特征向量时，可以使得J(W)最大。因为中c个秩为1或0的矩阵相加，而且其中只有c-1个矩阵是相互独立的，所以的秩最多为c-1，所以最多只有c-1个特征向量是非零的。

# 四、分类器设计

决策过程中，首先通过训练样本估计概率密度函数，再用统计决策进行类别判定，这种方法称作基于样本的两步贝叶斯决策。

在分类过程中，我选用了最小错误率贝叶斯决策。

以二分类问题为例说明最小错误率贝叶斯决策。在样本x上错误的概率为

( 4.1 )

错误率定义为所有服从同样分布的独立样本上错误概率的期望，即

( 4.2 )

由此可知最小错误率就是求解一种决策规则，使得上式最小化。由于对所有x，P(e|x)≥0，p(x)≥0，所以上式等价于对所有x最小化P(e|x)。而根据上上式可知，使错误率最小的决策就是使后验概率最大的决策，因此，对于两类问题，得到如下决策规则：

如果，则x；反之，则x

其中，后验概率用贝叶斯公式求得

( 4.3 )

其中先验概率和类条件密度都已知。在上式中，因为两类分母相同，所以实际决策时实际上只需要比较分子，即

若，则x

对于多类问题的决策思路，与两类问题的决策是类似的。

贝叶斯决策的基础是概率密度函数的估计，概率密度函数的估计方法又分为两大类：参数估计与非参数估计。本次实验中我采用的方法是正态分布下的最大似然估计。对于多元正态分布的均值和方差的最大似然估计推导过程不再赘述，这里直接给出公式：

( 4.4 )

( 4.5 )

多元正态分布公式如下：

( 4.6 )

其中||表示协方差矩阵的行列式。

在这里值得一提的是最大似然估计量是平方误差一致和简单一致估计量，但不一定都是无偏估计量。譬如这里是无偏估计量，但就不是无偏的。

# 五、评价方法

准确率的定义是：对于给定的测试数据集，分类器正确分类的样本数与总样本数之比。

ROC是Receiver Operating Characteristic的缩写。ROC曲线常被用来评价一个二值分类器的优劣，其横坐标为假阳性率（false positive rate,FPR），纵坐标为真阳性率（true positive rate,TPR）。下图详细说明了FPR和TPR是如何定义的。[4]

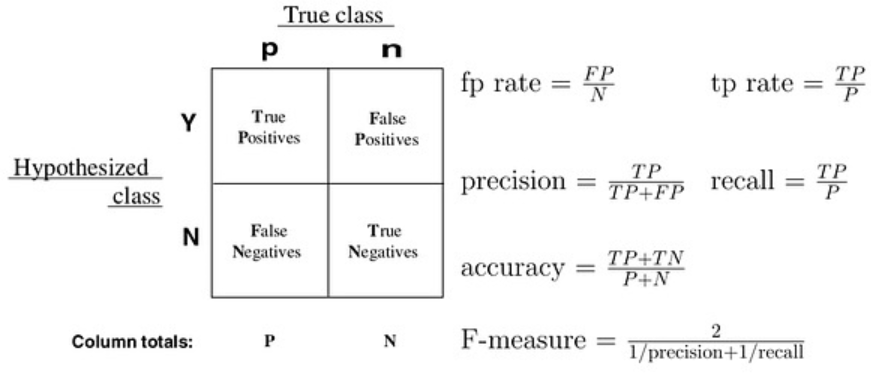


图 5.1 FPR与TPR定义

下图是一个ROC曲线的示例。

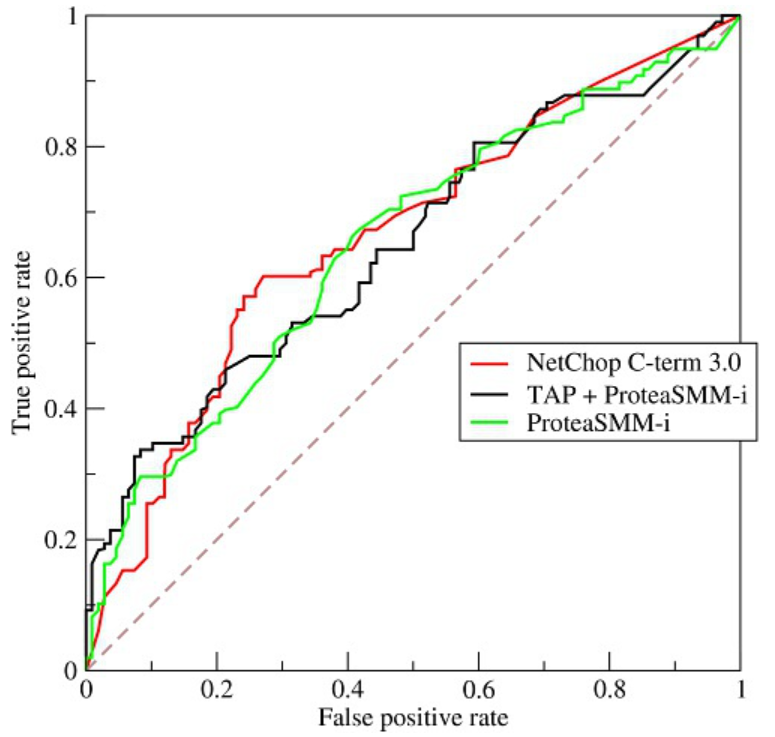


图 5.2 ROC曲线示例

图中，点（0,1），即FPR=0，TPR=1，这意味着FN(false negative)=0，并且FP（false positive）=0，这表明分类器将所有样本都正确分类；点（1,0），即FPR=1，TPR=0，表明这个分类器将所有样本错误分类。第三个点，(0,0)，即FPR=TPR=0，即FP（false positive）=TP（true positive）=0，可以发现该分类器预测所有的样本都为负样本（negative）。类似的，第四个点（1,1），分类器实际上预测所有的样本都为正样本。经过以上的分析，我们可以断言，ROC曲线越接近左上角，该分类器的性能越好。而ROC曲线图中的虚线y=x上的点。这条对角线上的点其实表示的是一个采用随机猜测策略的分类器的结果，例如(0.5,0.5)，表示该分类器随机对于一半的样本猜测其为正样本，另外一半的样本为负样本。

要绘制ROC曲线，需要一系列FPR和TPR的值；要得到这一系列的值，就需要调整决策的阈值；每确定一个阈值就决定了决策的真/假阳性率，对应曲线上的一个点。如何理解决策阈值？分类器的一个重要功能是“概率输出”，即表示分类器认为某个样本具有多大的概率属于正样本（或负样本）。假如我们已经得到了所有样本的概率输出（属于正样本的概率），如何调整决策阈值？根据每个测试样本属于正样本的概率值从大到小排序。下图是一个示例，图中共有20个测试样本，“Class”一栏表示每个测试样本真正的标签（p表示正样本，n表示负样本），“Score”表示每个测试样本属于正样本的概率。

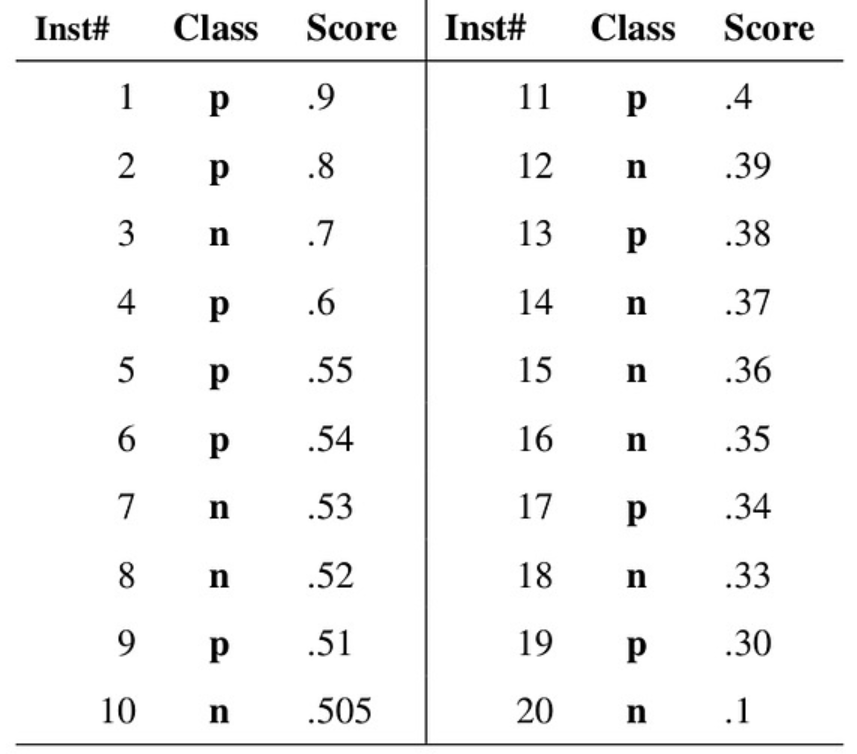


图 5.3 样本示例

接下来，我们从高到低，依次将“Score”值作为阈值threshold，当测试样本属于正样本的概率大于或等于这个threshold时，我们认为它为正样本，否则为负样本。举例来说，对于图中的第4个样本，其“Score”值为0.6，那么样本1，2，3，4都被认为是正样本，因为它们的“Score”值都大于等于0.6，而其他样本则都认为是负样本。每次选取一个不同的threshold，我们就可以得到一组FPR和TPR，即ROC曲线上的一点。这样一来，我们一共得到了20组FPR和TPR的值，将它们画在ROC曲线的结果如下图：

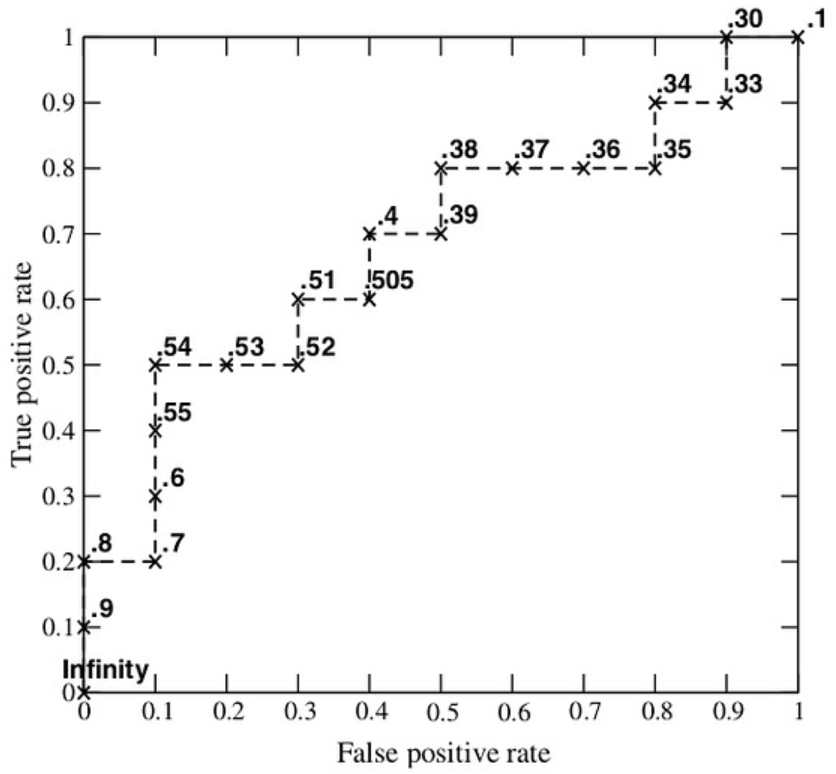


图 5.4 绘制ROC曲线

当我们将阈值设置为1和0时，分别可以得到ROC曲线上的(0,0)和(1,1)两个点。将这些(FPR,TPR)对连接起来，就得到了ROC曲线。当阈值取值越多，ROC曲线越平滑。

采用ROC曲线作为评价标准是因为ROC曲线有个很好的特性：当测试集中的正负样本的分布变化的时候，ROC曲线能够保持不变。在实际的数据集中经常会出现类不平衡现象，即负样本比正样本多很多（或者相反），而且测试数据中的正负样本的分布也可能随着时间变化。下图是ROC曲线和Precision-Recall曲线的对比：

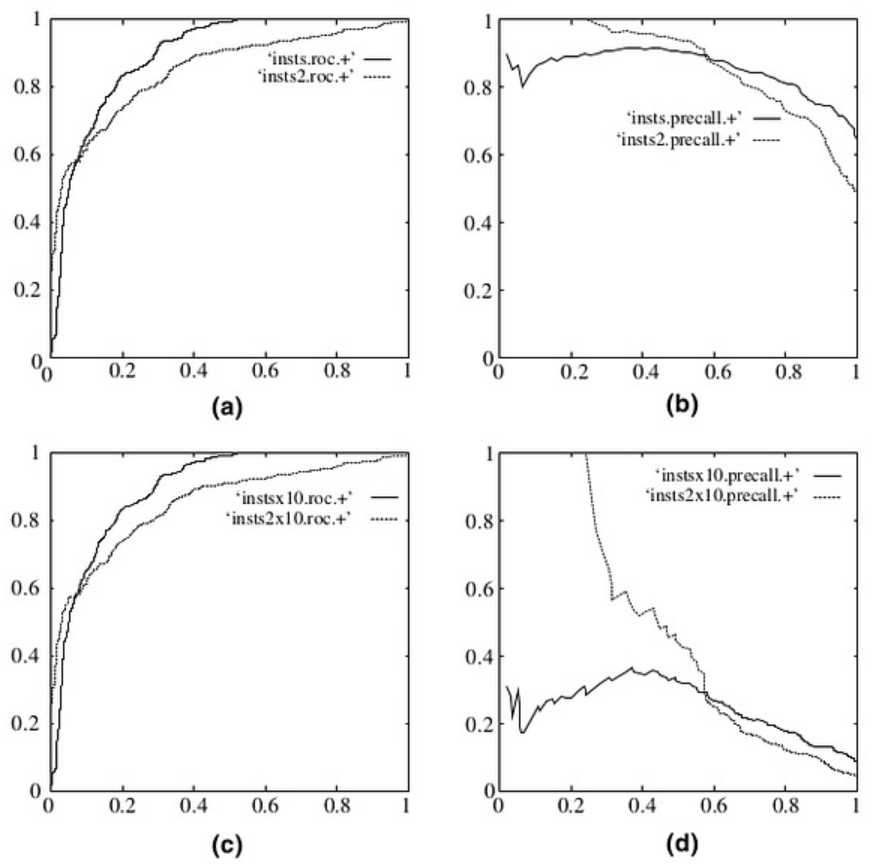


图 5.5 ROC曲线与Prision-Recall曲线对比

在上图中，(a)和(c)为ROC曲线，(b)和(d)为Precision-Recall曲线。(a)和(b)展示的是分类其在原始测试集（正负样本分布平衡）的结果，(c)和(d)是将测试集中负样本的数量增加到原来的10倍后，分类器的结果。可以明显的看出，ROC曲线基本保持原貌，而Precision-Recall曲线则变化较大。

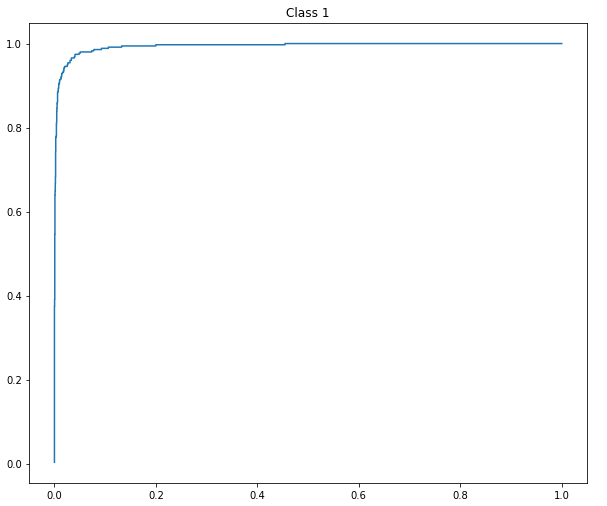
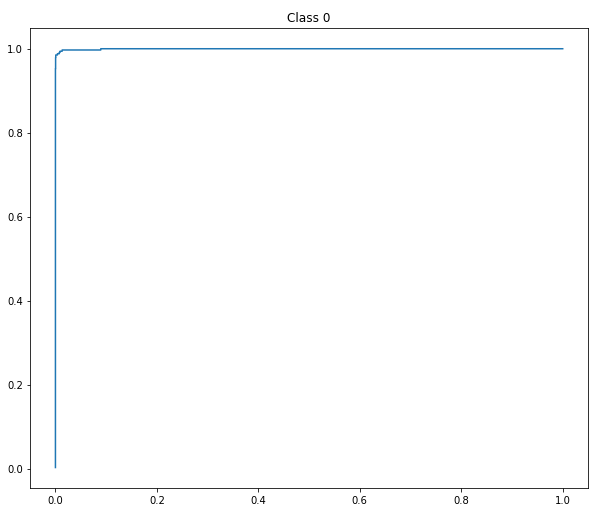
# 六、具体实现

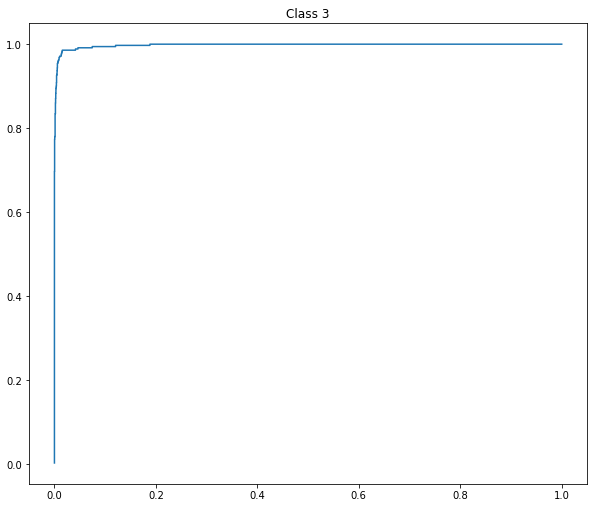
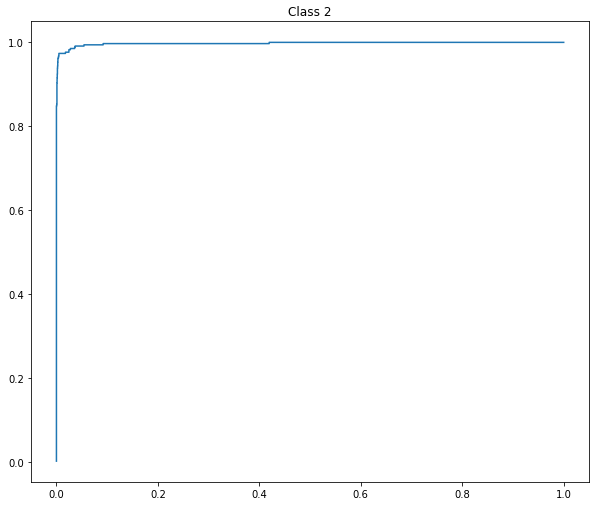
读入训练集数据后，以给定比例将其分为训练数据与验证数据（代码中训练数据与验证数据之比为9：1）；训练数据用于估计参数，在利用LDA方法对特征进行降维后（代码中64维特征被降至9维），分别得出每种手写数字特征在正态分布下参数(均值与协方差)的最大似然估计，再根据已知的先验概率与类条件密度概率得出每个样本属于每种类别的后验概率，样本属于某个类别的后验概率最大时，就预测这个样本属于这一类别。

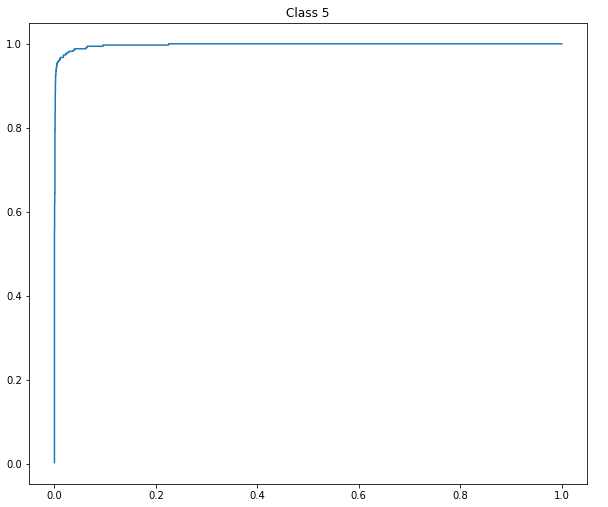
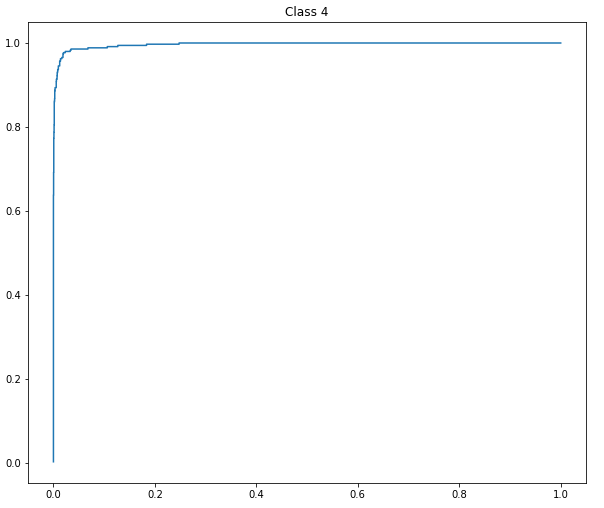
ROC曲线是针对二值分类器的，而手写数字光学识别则是一个多分类问题，故代码中针对每种类别分别绘制了其ROC曲线。

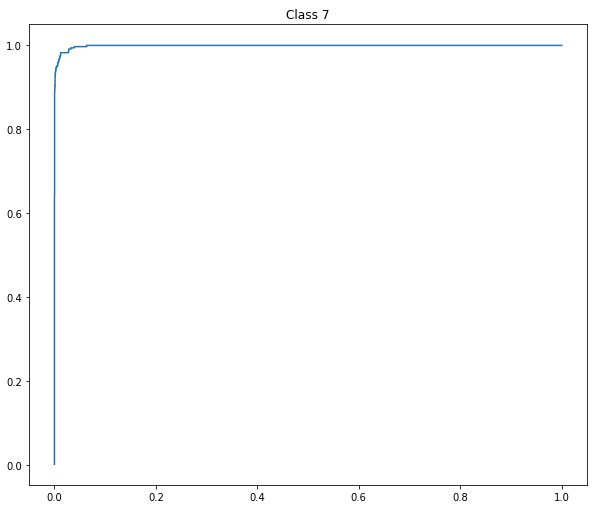
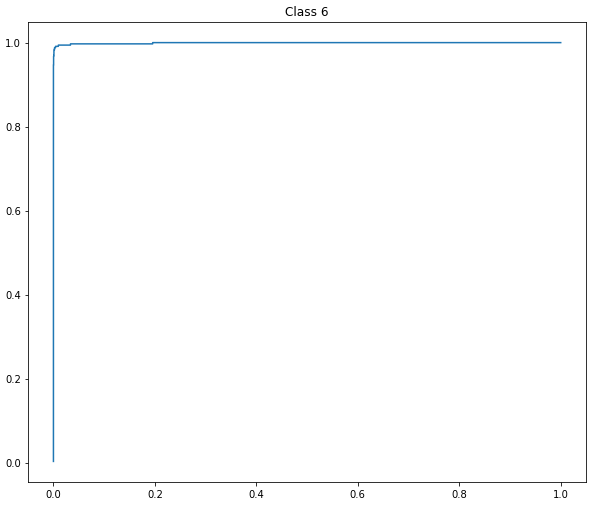
# 七、测试结果

训练集中训练部分数据的准确率：0.9726505673552。手写数字0到9的识别ROC曲线分别如下图所示。









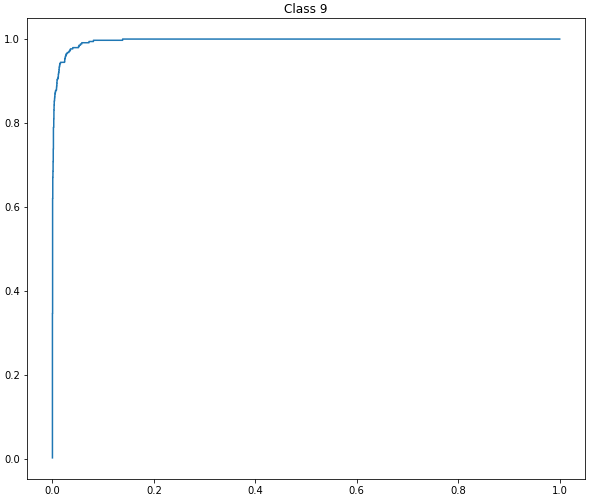
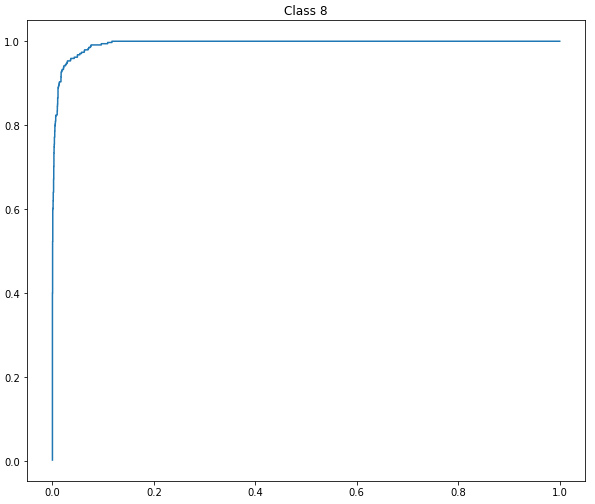
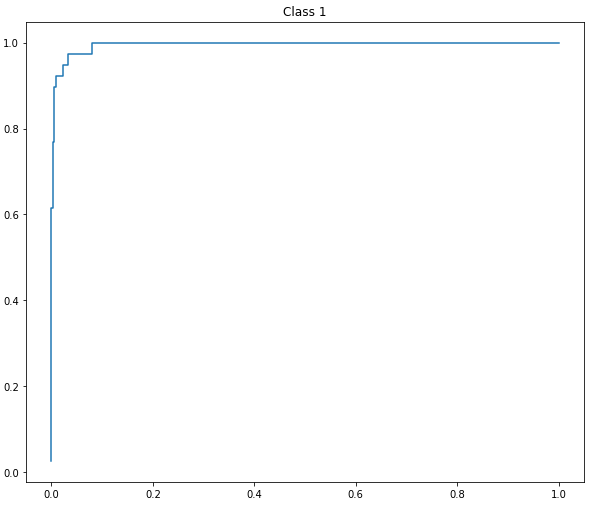
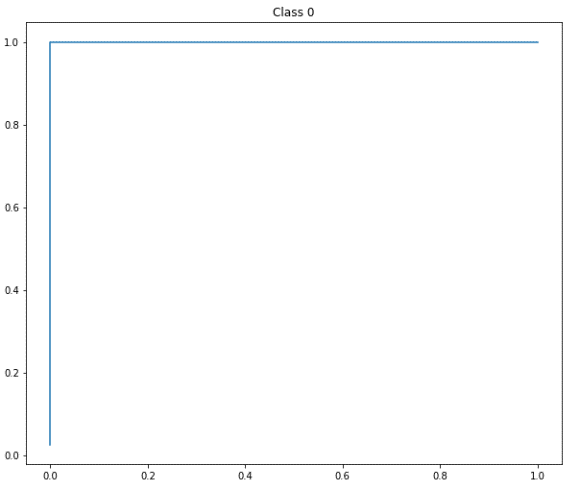
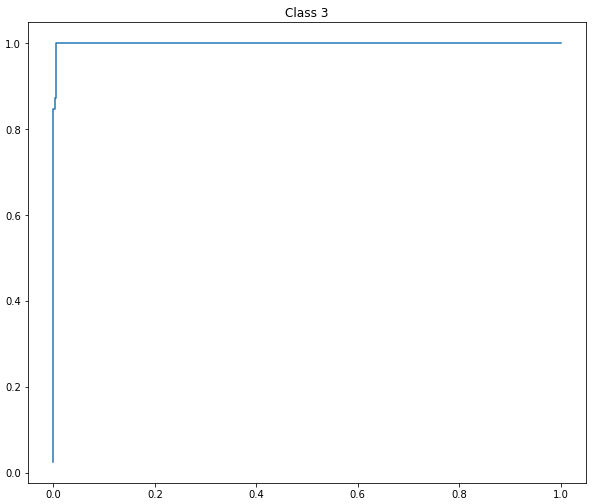
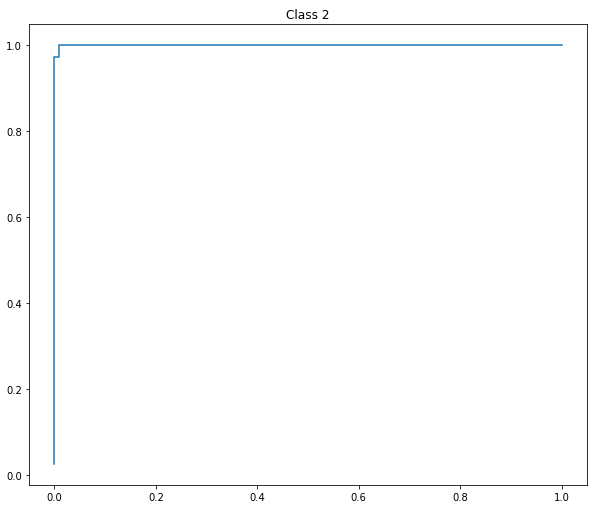
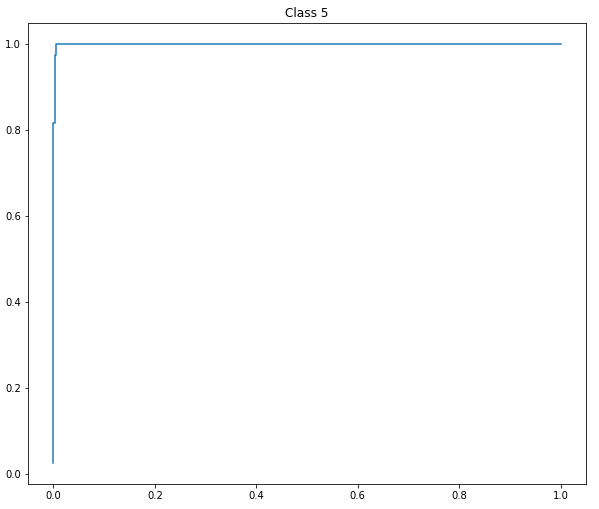
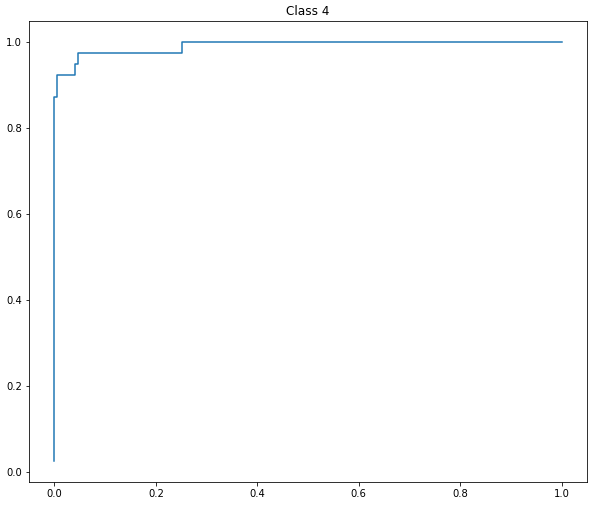


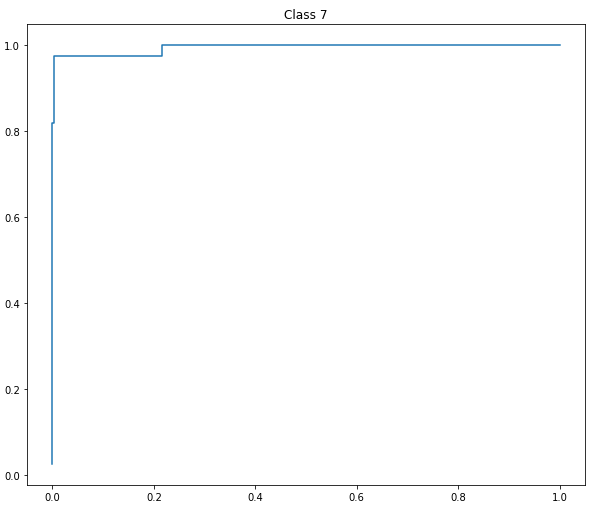
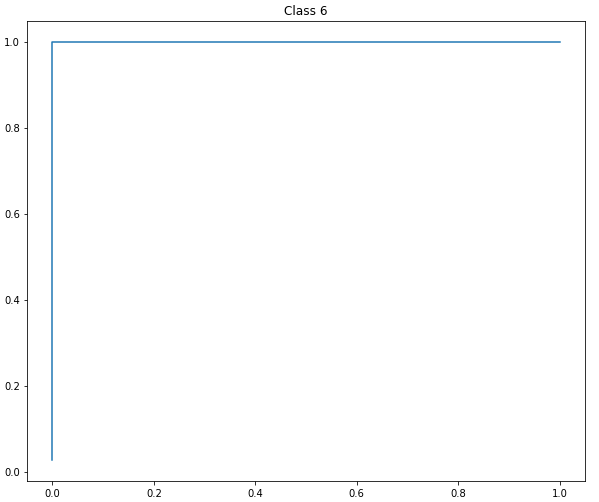
图 6.1 训练集训练数据ROC曲线

训练集中验证部分数据的准确率：0.9818181818181。手写数字0到9的识别ROC曲线分别如下图所示。









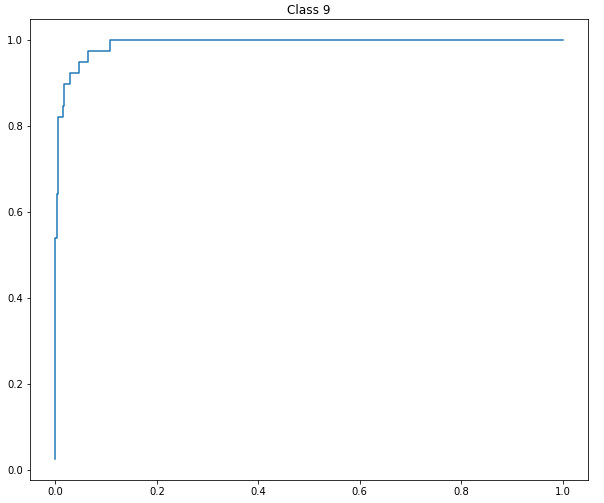
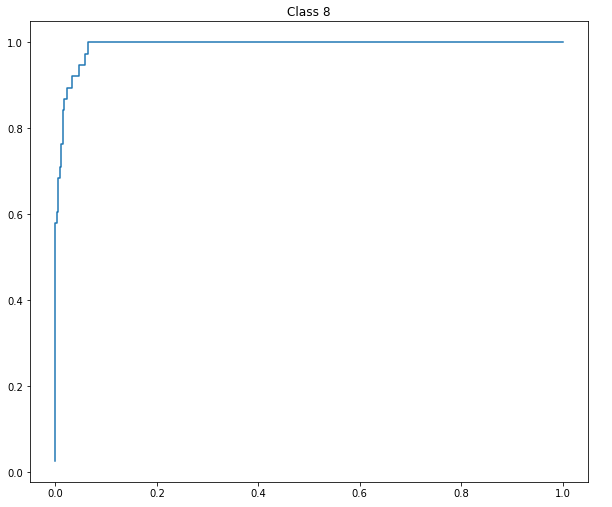
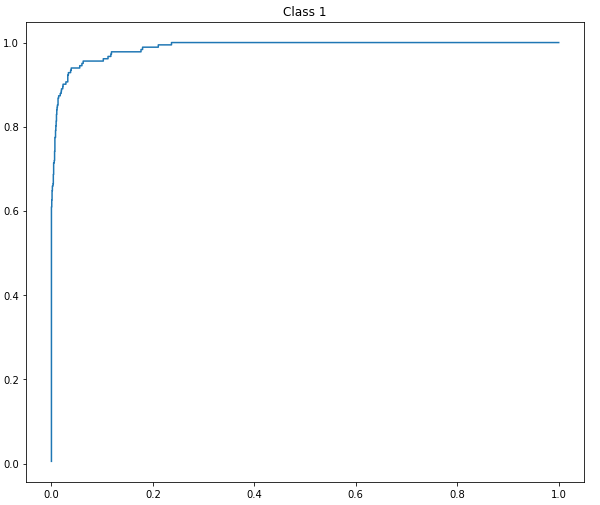
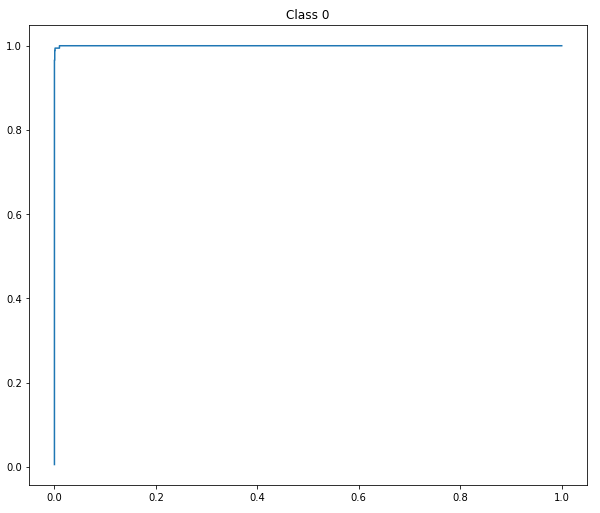
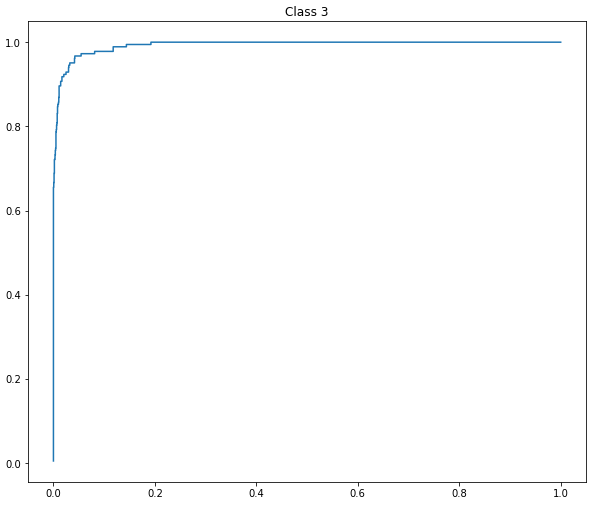
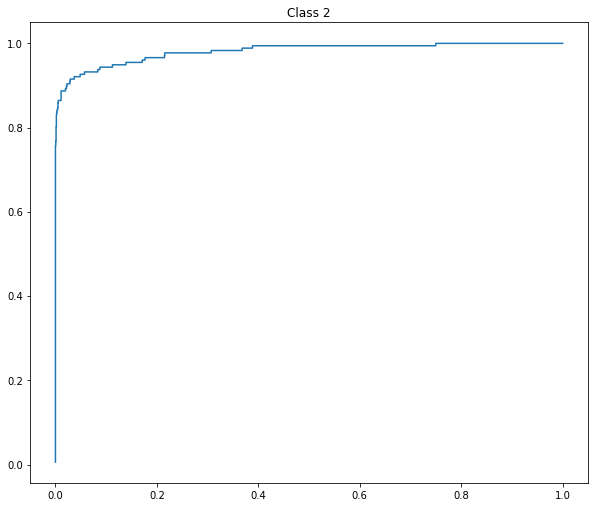
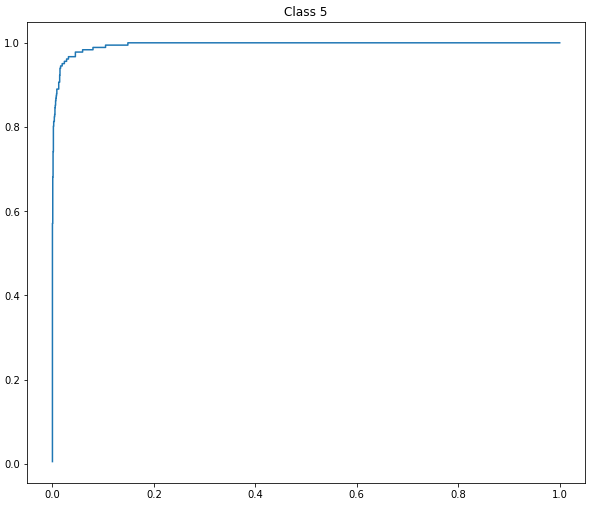
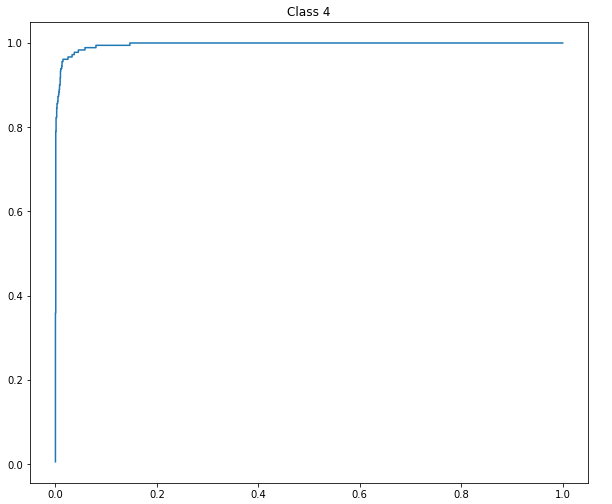


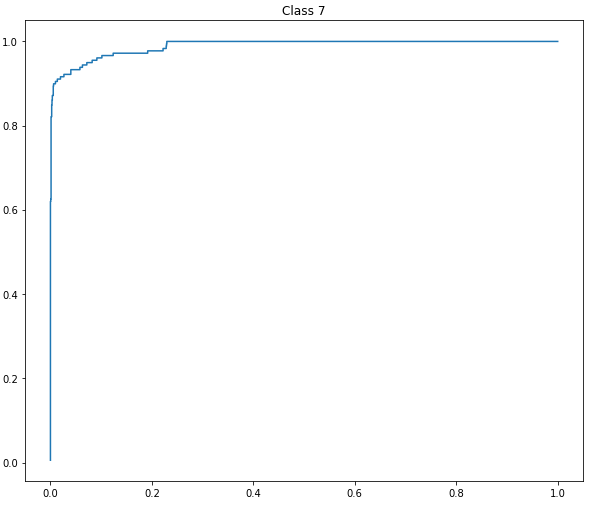
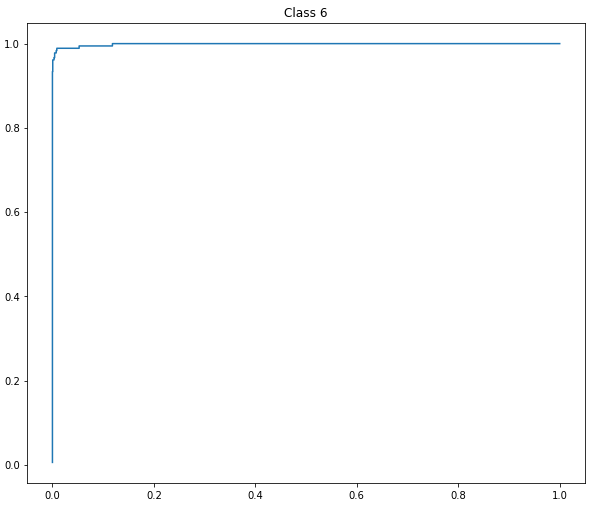
图6.2 训练集验证数据ROC曲线

测试集数据的识别准确率：0.9404231625835。手写数字0到9的识别ROC曲线分别如下图所示。









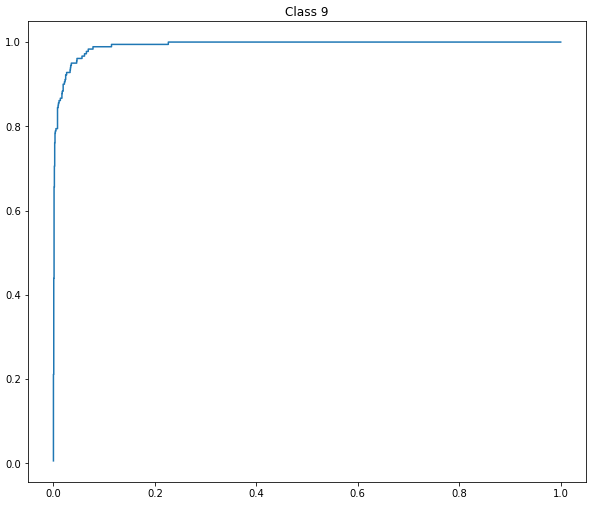
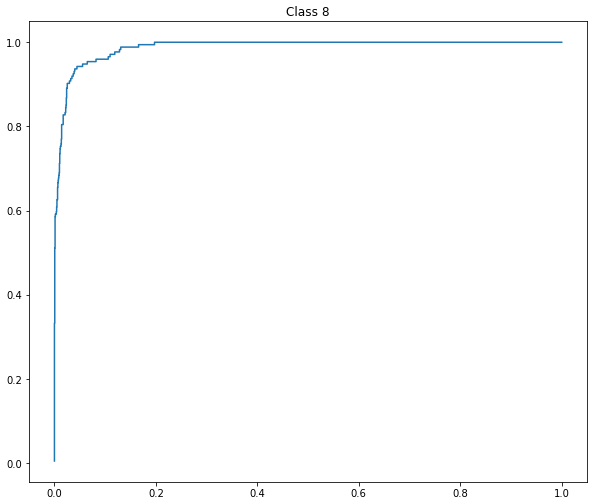


图6.3 测试集数据ROC曲线

将数据投影到二维上后可视化的效果如下图所示。可以直观感受到训练集与测试集中不同类别的分布是相近的。

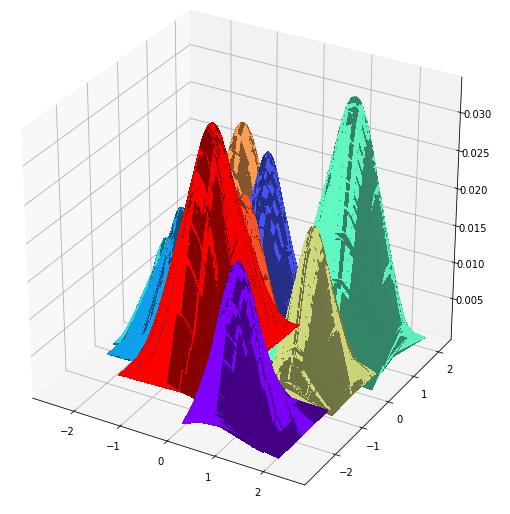
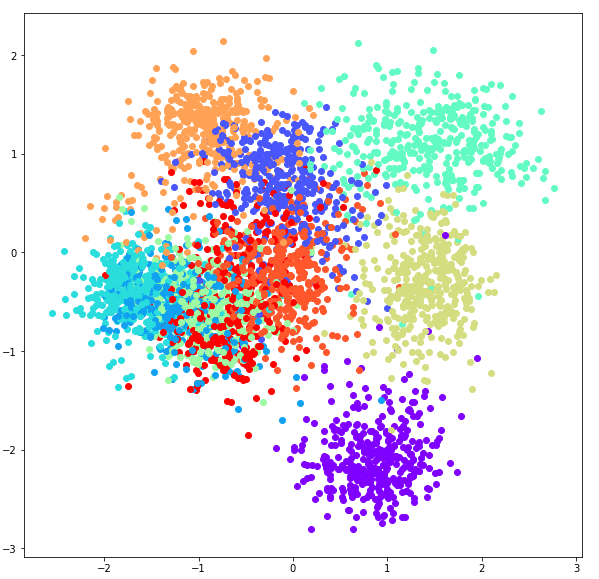


图6.4 训练集训练数据二维及三维平面投影

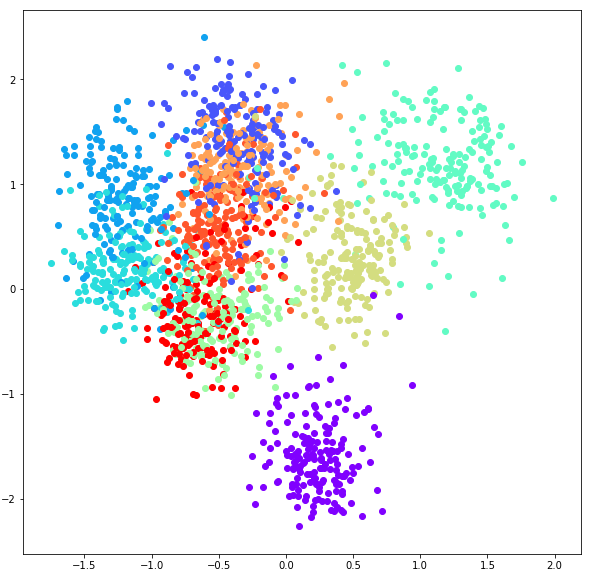
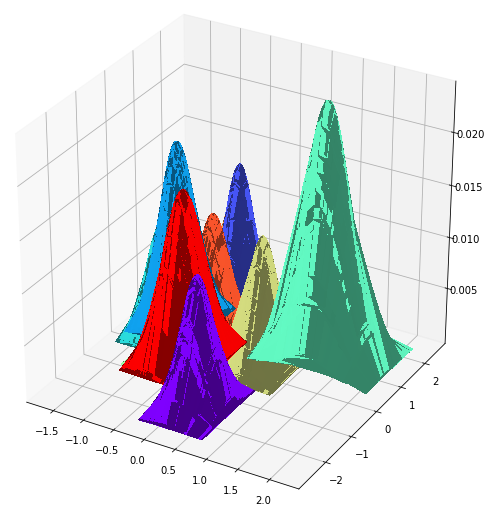


图 6.5 测试集数据二维及三维平面投影

# 八、结论

由准确率及ROC曲线可以看出，在将特征维数由64维降至9维后，样本的分类效果是可以接受的。但同时也可以发现，分类器在训练集的验证数据上表现较好，而在完全独立的测试集上分类效果有所下降。原因在于训练集的训练部分数据与验证部分数据是由同一批受试者贡献的数据，而测试集的数据则是另外不同的13个人的手写数字，他们的手写数字特征是有差异的，故而在测试集上分类效果有所下降。

由于时间原因，我没有进一步去学习实现更加有难度的算法，比如SVM，感知器算法等等。这是一个遗憾。在接下来进一步学习的过程中，我会继续从原理层级学习并实现这些经典的算法。

# 参考文献

1. E, Alpaydin, C, Kaynak. Optical Recognition of Handwritten Digits[DB/OL]. https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits.
2. 张学工. 模式识别[M]. 北京:张学工, 2010.
3. 维基百科. 费雪线性判别[EB/OL]. https://zh.wikipedia.org/wiki/費雪線性判別.
4. 孔明. ROC和AUC介绍以及如何计算AUC[EB/OL]. http://alexkong.net/2013/06/introduction-to-auc-and-roc/.

# 附录

import sys, itertools

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.cm as cm

from mpl\_toolkits.mplot3d import Axes3D

import random

class LDA():

def \_\_init\_\_(self, data, testingdata, num\_dims = 1, convert\_data = 0,

percentile = 50, threshold = 0, labelcol = -1, split\_ratio = 0.9):

'''

类的组成

--------------------------------

data : 训练集

testingdata : 测试集

num\_dims : 降维后特征维数

convert\_data : 是否将数据标签转化为二分类标签

percentile : 如果要将数据标签转化为二分类标签，转化比例

threshold : 标记是进行二分类还是多分类

labelcol : 数据的标签列

split\_ratio : 训练集中分割出验证数据的比例

'''

self.data = data

self.num\_dims = num\_dims

self.convert\_data = convert\_data

self.percentile = percentile

self.threshold = threshold

self.labelcol = labelcol

self.split\_ratio = split\_ratio

self.testingdata = testingdata

if (self.convert\_data):

self.data = self.to\_categorical(self.data, self.percentile)

'''

将数据根据比例转化为二分类

'''

def to\_categorical(self, data, percentile):

fraction = percentile / 100.0

# partition data based on percentile of label column

med = data.ix[:, self.labelcol].quantile(fraction)

for i in range(data.shape[0]):

if (data.ix[i, self.labelcol] >= med):

data.ix[i, self.labelcol] = 1

else:

data.ix[i, self.labelcol] = 0

return data

'''

删除数据特定列

'''

def drop\_col(self, data, col):

return data.drop(data.columns[[col]], axis = 1)

'''

进行LDA降维的主要函数

'''

def fit(self):

# 估计LDA参数

def estimate\_params(data):

# 根据类标签将数据分组

grouped = data.groupby(self.data.ix[:,self.labelcol])

# 计算每一类的特征均值

means = {}

for c in self.classes:

means[c] = np.array(self.drop\_col(self.classwise[c], self.labelcol).mean(axis = 0))

# 数据的整体均值

overall\_mean = np.array(self.drop\_col(data, self.labelcol).mean(axis = 0))

# 类间离散度矩阵

# S\_B = \sigma{N\_i (m\_i - m) (m\_i - m).T}

S\_B = np.zeros((data.shape[1] - 1, data.shape[1] - 1))

for c in means.keys():

S\_B += np.multiply(len(self.classwise[c]),

np.outer((means[c] - overall\_mean),

(means[c] - overall\_mean)))

# 类内离散度矩阵

# S\_W = \sigma{S\_i}

# S\_i = \sigma{(x - m\_i) (x - m\_i).T}

S\_W = np.zeros(S\_B.shape)

for c in self.classes:

tmp = np.subtract(self.drop\_col(self.classwise[c], self.labelcol).T, np.expand\_dims(means[c], axis=1))

S\_W = np.add(np.dot(tmp, tmp.T), S\_W)

# 计算inv(S\_W).S\_B的特征值、特征向量

mat = np.dot(np.linalg.pinv(S\_W), S\_B)

eigvals, eigvecs = np.linalg.eig(mat)

eiglist = [(eigvals[i], eigvecs[:, i]) for i in range(len(eigvals))]

# 根据特征值将特征向量由大到小排序

eiglist = sorted(eiglist, key = lambda x : x[0], reverse = True)

# 根据降维后的维数取出前num\_dim个特征向量

w = np.array([eiglist[i][1] for i in range(self.num\_dims)])

self.w = w

self.means = means

return

# 分离训练数据与验证数据

traindata = []

testdata = []

# 根据类别标签将数据分组

grouped = data.groupby(self.data.ix[:,self.labelcol])

self.classes = [c for c in grouped.groups.keys()]

self.classwise = {}

for c in self.classes:

self.classwise[c] = grouped.get\_group(c)

rows = random.sample(list(self.classwise[c].index),

int(self.classwise[c].shape[0] \*

self.split\_ratio))

traindata.append(self.classwise[c].ix[rows])

testdata.append(self.classwise[c].drop(rows))

traindata = pd.concat(traindata)

testdata = pd.concat(testdata)

# 估计LDA参数

estimate\_params(traindata)

# 对验证数据分类

# 二分类

if (self.threshold):

self.calculate\_threshold()

# append the training and test error rates for this iteration

trainerror = self.calculate\_score(traindata) / float(traindata.shape[0])

testerror = self.calculate\_score(testdata) / float(testdata.shape[0])

# 多分类

else:

self.gaussian\_modeling()

# 计算错误率

trainerror = self.calculate\_score\_gaussian(traindata) / float(traindata.shape[0])

testerror = self.calculate\_score\_gaussian(testdata) / float(testdata.shape[0])

testingerror = self.calculate\_score\_gaussian(self.testingdata) / float(self.testingdata.shape[0])

return trainerror, testerror, testingerror

'''

根据类别标签均值进行二分类

'''

def calculate\_threshold(self):

tot = 0

for c in self.means.keys():

tot += np.dot(self.w, self.means[c])

self.w0 = 0.5 \* tot

# 大于均值为第一类，小于均值为第二类

c1 = self.means.keys()[0]

c2 = self.means.keys()[1]

mu1 = np.dot(self.w, self.means[c1])

if (mu1 >= self.w0):

self.c1 = 'ge'

else:

self.c1 = 'l'

'''

二分类

'''

def calculate\_score(self, data):

inputs = self.drop\_col(data, self.labelcol)

# project the inputs

proj = np.dot(self.w, inputs.T).T

# assign the predicted class

c1 = self.means.keys()[0]

c2 = self.means.keys()[1]

if (self.c1 == 'ge'):

proj = [c1 if proj[i] >= self.w0 else c2 for i in range(len(proj))]

else:

proj = [c1 if proj[i] < self.w0 else c2 for i in range(len(proj))]

# calculate the number of errors made

errors = (proj != data.ix[:, self.labelcol])

return sum(errors)

'''

计算每个类别所服从的高斯分布密度函数

每个类别的均值、协方差矩阵、先验概率

'''

def gaussian\_modeling(self):

self.priors = {}

self.gaussian\_means = {}

self.gaussian\_cov = {}

for c in self.means.keys():

inputs = self.drop\_col(self.classwise[c], self.labelcol)

proj = np.dot(self.w, inputs.T).T

self.priors[c] = inputs.shape[0] / float(self.data.shape[0])

self.gaussian\_means[c] = np.mean(proj, axis = 0)

self.gaussian\_cov[c] = np.cov(proj, rowvar=False)

'''

给出均值、协方差矩阵，计算相应点的类条件概率

'''

def pdf(self, point, mean, cov):

cons = 1./((2\*np.pi)\*\*(len(point)/2.)\*np.linalg.det(cov)\*\*(-0.5))

return cons\*np.exp(-np.dot(np.dot((point-mean),np.linalg.inv(cov)),(point-mean).T)/2.)

'''

高斯模型分类错误率，绘制ROC曲线

'''

def calculate\_score\_gaussian(self, data):

classes = sorted(list(self.means.keys()))

inputs = self.drop\_col(data, self.labelcol)

# project the inputs

proj = np.dot(self.w, inputs.T).T

# calculate the likelihoods for each class based on the gaussian models

likelihoods = np.array([[self.priors[c] \* self.pdf([x[ind] for ind in

range(len(x))], self.gaussian\_means[c],

self.gaussian\_cov[c]) for c in

classes] for x in proj])

# assign prediction labels based on the highest probability

labels = np.argmax(likelihoods, axis = 1)

errors = np.sum(labels != data.ix[:, self.labelcol])

confusion\_matrix = np.zeros((len(classes), len(classes)))

for i,j in zip(labels, data.ix[:, self.labelcol]):

confusion\_matrix[i][j] += 1

class\_likelihoods = np.zeros(len(data))

TP = np.zeros(likelihoods.shape)

FP = np.zeros(likelihoods.shape)

FN = np.zeros(likelihoods.shape)

TN = np.zeros(likelihoods.shape)

#FPR = np.zeros(len(classes))

#TPR = np.zeros(len(classes))

FPR\_func = lambda FP,TN:FP/(FP+TN)

TPR\_func = lambda TP,FN:TP/(TP+FN)

f, axs = plt.subplots(len(classes),1,figsize=(10,10\*len(classes)))

for c in classes:

class\_likelihoods = sorted(list(enumerate(likelihoods[:,c])), key = lambda item:item[1], reverse = True)

for i in range(len(class\_likelihoods)):

TP[i, c] = list(data.iloc[[item[0] for item in class\_likelihoods[0:i+1]], self.labelcol]).count(c)

FP[i, c] = i + 1 - TP[i, c]

FN[i, c] = list(data.iloc[[item[0] for item in class\_likelihoods[i+1:]], self.labelcol]).count(c)

TN[i, c] = len(class\_likelihoods) - i - 1 - FN[i, c]

FPR = list(map(FPR\_func, FP[:,c],TN[:,c]))

TPR = list(map(TPR\_func, TP[:,c],FN[:,c]))

axs[c].set\_title('Class {}'.format(c))

axs[c].plot(FPR,TPR)

return errors

def plot\_bivariate\_gaussians(self):

classes = list(self.means.keys())

colors = cm.rainbow(np.linspace(0, 1, len(classes)))

plotlabels = {classes[c] : colors[c] for c in range(len(classes))}

fig = plt.figure(figsize=(10,10))

ax3D = fig.add\_subplot(111, projection='3d')

for c in self.means.keys():

data = np.random.multivariate\_normal(self.gaussian\_means[c],

self.gaussian\_cov[c], size=100)

pdf = np.zeros(data.shape[0])

cons = 1./((2\*np.pi)\*\*(data.shape[1]/2.)\*np.linalg.det(self.gaussian\_cov[c])\*\*(-0.5))

X, Y = np.meshgrid(data.T[0], data.T[1])

def pdf(point):

return cons\*np.exp(-np.dot(np.dot((point-self.gaussian\_means[c]),np.linalg.inv(self.gaussian\_cov[c])),(point-self.gaussian\_means[c]).T)/2.)

zs = np.array([pdf(np.array(ponit)) for ponit in zip(np.ravel(X),

np.ravel(Y))])

Z = zs.reshape(X.shape)

surf = ax3D.plot\_surface(X, Y, Z, rstride=1, cstride=1,

color=plotlabels[c], linewidth=0,

antialiased=False)

plt.show()

def plot\_proj\_1D(self, data):

classes = list(self.means.keys())

colors = cm.rainbow(np.linspace(0, 1, len(classes)))

plotlabels = {classes[c] : colors[c] for c in range(len(classes))}

fig = plt.figure()

for i, row in data.iterrows():

proj = np.dot(self.w, row[:self.labelcol])

plt.scatter(proj, np.random.normal(0,1,1)+0, color =

plotlabels[row[self.labelcol]])

plt.show()

def plot\_proj\_2D(self, data):

classes = list(self.means.keys())

colors = cm.rainbow(np.linspace(0, 1, len(classes)))

plotlabels = {classes[c] : colors[c] for c in range(len(classes))}

fig = plt.figure(figsize=(10,10))

for i, row in data.iterrows():

proj = np.dot(self.w, row[:self.labelcol])

plt.scatter(proj[0], proj[1], color =

plotlabels[row[self.labelcol]])

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

data = pd.read\_csv('train.csv')

testingdata = pd.read\_csv('test.csv')

labelcol = int(-1)

lda = LDA(data, testingdata, num\_dims=9, convert\_data=0, threshold=0, labelcol=labelcol)

trainerror, testerror, testingerror = lda.fit()

print(trainerror)

print(testerror)

print(testingerror)

#print(verifyLDA(data, data, labelcol))

#lda.plot\_proj\_2D(data)

#lda.plot\_bivariate\_gaussians()