MS328 Assignment4

周李韬 5180309104O7

2020 年 5 月 2 日

PCA 方法

找一个有意思的真实数据，尝试基于所学的 PCA 方法对数据完成降维，并探索降维的效果。

1 问题分析

1.1 问题叙述

本次作业中，我们将会使用 MNIST 数据集 (http://yann.lecun.com/exdb/mnist/)，该数据集包含 70000 张规格较小的手写数字图片，由美国的高中生和美国人口调查局的职员手写而成。由于图片含有较多维数的特征，直接进行分类将产生维数灾难，我们会利用 PCA 方法对数据进行降维。对比使用 PCA 前后分类效果的差异，同时我们还会通过可视化的方法展现降维后数据的分布。

首先我们从 (https://osf.io/jda6s/) 下载 mat 格式的数据并导入，样本维数为 784（28×28），每一维度的值都是 $0 \sim 255$ 的一个灰度值。标签是 $0 \sim 9$ 的整数。

[1]:
```
import numpy as np
from scipy.io import loadmat
mnist = loadmat('mnist-original.mat')
images = np.transpose(np.array(mnist["data"],dtype=np.uint8))
labels = np.transpose(np.array(mnist["label"],dtype=np.uint8))
print(images,images.shape)
print(labels,labels.shape)
```

```
[[0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
 …
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
```

```
 [0 0 0 … 0 0 0]] (70000, 784)
[[0]
 [0]
 [0]
 …
 [9]
 [9]
 [9]] (70000, 1)
```

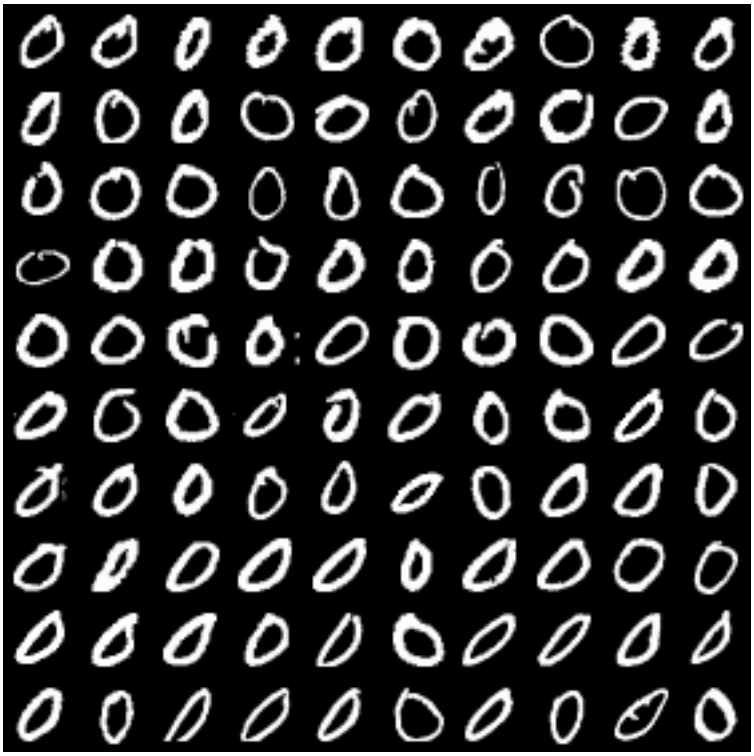
我们可以将灰度数值转化为图像，获得有关数据集的直观认识。

[2]:
```
from PIL import Image

def array_to_img(array):
    array=array
    new_img=Image.fromarray(array.astype(np.uint8))
    return new_img

def comb_imgs(origin_imgs, col, row, each_width, each_height, new_type):
    new_img = Image.new(new_type, (col* each_width, row* each_height))
    for i in range(len(origin_imgs)):
        each_img = array_to_img(np.array(origin_imgs[i]).reshape(each_width,␣
 ↪each_width))
        new_img.paste(each_img, (int((i % col) * each_width), int((i // col) *␣
 ↪each_width)))
    return new_img

test_imgs=comb_imgs(images[:100], 10, 10, 28, 28, 'L')
display(test_imgs)
```



2 实验 1：原始数据的 SVM 分类

由于原始数据维数较大，我们采用随机梯度下降法的 SVM 进行训练和分类。对标签我们做二分类处理，仅识别数字 5。我们采用留出法对模型进行简单测试，我们调用 sklearn 的方法将数据集随机分为 75% 的训练集和 25% 的测试集。

[37]:
```
labels_5 = (labels == 5)  # 对于所有 5 为真，对于所有其他数字为假

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(images, np.ravel(labels_5),␣
↪test_size=0.25)
from sklearn import linear_model
clf = linear_model.SGDClassifier(max_iter=1000, tol=1e-3)
clf.fit(X_train, y_train)
```

[37]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
```

```
                   l1_ratio=0.15, learning_rate='optimal', loss='hinge',
                   max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
                   power_t=0.5, random_state=None, shuffle=True, tol=0.001,
                   validation_fraction=0.1, verbose=0, warm_start=False)
```

[38]:
```
def test_analysis(y_test,result):
    result = np.c_[result,y_test]
    TP, FP, TN, FN = 0, 0, 0, 0
    for i in range(result.shape[0]):
        # print (result[i])
        if (result[i][0] == 0 and result[i][1] == 0):
            TN += 1
        if (result[i][0] == 0 and result[i][1] == 1):
            FN += 1
        if (result[i][0] == 1 and result[i][1] == 1):
            TP += 1
        if (result[i][0] == 1 and result[i][1] == 0):
            FP += 1
    print ("TN: ",TN,"  FN: ",FN,"  TP: ",TP,"  FP: ",FP)
    print ("查准率 P: ", TP/(TP+FP))
    print ("查全率 R: ", TP/(TP+FN))
    print ("F1: ", 2*TP/(2*TP+FN+FP))
    return (2*TP/(2*TP+FN+FP))
```

[39]:
```
test_result = clf.predict(X_test)
X_pred = (np.sign(test_result - 0.5) + 1)/2
X_pred.astype(np.int64)
test_analysis(X_pred,y_test)
```

```
TN:  15764   FN:  145   TP:  1150   FP:  441
查准率 P:  0.7228158390949089
查全率 R:  0.888030888030888
F1:  0.796950796950797
```

[39]:
```
0.796950796950797
```

3 实验 2：PCA 处理后的模型对比

我们调用 sklearn 的 PCA 方法对原始数据做主成分分析。采用相同配置的模型进行训练。

[40]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=400)
pca.fit(images)
new_images=pca.transform(images)
```

[41]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(new_images, np.
 ↪ravel(labels_5), test_size=0.25)
new_clf = linear_model.SGDClassifier(max_iter=1000, tol=1e-3)
new_clf.fit(X_train, y_train)
test_result = new_clf.predict(X_test)
X_pred = (np.sign(test_result - 0.5) + 1)/2
X_pred.astype(np.int64)
test_analysis(X_pred,y_test)
```

```
TN:  15755   FN:  156   TP:  1238   FP:  351
查准率 P:  0.7791063561988673
查全率 R:  0.8880918220946915
F1:  0.8300368756285619
```

[41]:
```
0.8300368756285619
```

我们发现提取 400 个特征后，F1 值有明显的提高，说明 PCA 可以去除样本中的噪音，提高模型的准确性。

进一步，我们探索 PCA 选取特征数目对模型效果的影响。我们通过迭代的方式，在相同的框架和条件下，进行多次训练和测试，得到如下结果。

[42]:
```
result = []
for n_comp in range(50,701,50):
    globals()['pca_'+str(n_comp)] = PCA(n_components=n_comp)
    globals()['pca_'+str(n_comp)] .fit(images)
    new_images=globals()['pca_'+str(n_comp)] .transform(images)
```

```
    X_train, X_test, y_train, y_test = train_test_split(new_images, np.
↪ravel(labels_5), test_size=0.25)
    globals()['clf_'+str(n_comp)] = linear_model.SGDClassifier(max_iter=1000,␣
↪tol=1e-3)
    globals()['clf_'+str(n_comp)].fit(X_train, y_train)
    test_result = globals()['clf_'+str(n_comp)].predict(X_test)
    X_pred = (np.sign(test_result - 0.5) + 1)/2
    X_pred.astype(np.int64)
    print("------- n_components = "+str(n_comp)+" ---------")
    result.append([n_comp,test_analysis(X_pred,y_test)])
```

```
------- n_components = 50 ---------
TN:  15662   FN:  260   TP:  1004   FP:  574
查准率 P:  0.6362484157160964
查全率 R:  0.7943037974683544
F1:  0.7065446868402533
------- n_components = 100 ---------
TN:  15623   FN:  282   TP:  1177   FP:  418
查准率 P:  0.7379310344827587
查全率 R:  0.8067169294037012
F1:  0.77079240340537
------- n_components = 150 ---------
TN:  15667   FN:  262   TP:  1185   FP:  386
查准率 P:  0.7542966263526416
查全率 R:  0.8189357290946786
F1:  0.7852882703777336
------- n_components = 200 ---------
TN:  15779   FN:  177   TP:  1222   FP:  322
查准率 P:  0.7914507772020726
查全率 R:  0.873481057898499
F1:  0.8304451240231057
------- n_components = 250 ---------
TN:  15692   FN:  196   TP:  1211   FP:  401
查准率 P:  0.7512406947890818
查全率 R:  0.8606965174129353
F1:  0.8022524014574363
```

------- n_components = 300 ---------
TN: 15674   FN: 224   TP: 1240   FP: 362
查准率 P: 0.7740324594257179
查全率 R: 0.8469945355191257
F1: 0.8088714938030006
------- n_components = 350 ---------
TN: 15745   FN: 202   TP: 1224   FP: 329
查准率 P: 0.7881519639407598
查全率 R: 0.8583450210378681
F1: 0.8217522658610272
------- n_components = 400 ---------
TN: 15737   FN: 207   TP: 1202   FP: 354
查准率 P: 0.7724935732647815
查全率 R: 0.8530872959545777
F1: 0.8107925801011805
------- n_components = 450 ---------
TN: 15655   FN: 253   TP: 1336   FP: 256
查准率 P: 0.8391959798994975
查全率 R: 0.8407803650094399
F1: 0.839987425337944
------- n_components = 500 ---------
TN: 15734   FN: 184   TP: 1277   FP: 305
查准率 P: 0.8072060682680152
查全率 R: 0.8740588637919233
F1: 0.8393033190930004
------- n_components = 550 ---------
TN: 15742   FN: 214   TP: 1258   FP: 286
查准率 P: 0.8147668393782384
查全率 R: 0.8546195652173914
F1: 0.8342175066312998
------- n_components = 600 ---------
TN: 15655   FN: 252   TP: 1326   FP: 267
查准率 P: 0.832391713747646
查全率 R: 0.8403041825095057
F1: 0.836329233680227
------- n_components = 650 ---------
TN: 15697   FN: 255   TP: 1256   FP: 292

查准率 P: 0.8113695090439277

查全率 R: 0.8312375909993381

F1: 0.8211833932657732

------- n_components = 700 ---------

TN: 15723 FN: 248 TP: 1232 FP: 297

查准率 P: 0.8057553956834532

查全率 R: 0.8324324324324325

F1: 0.8188767032236623

观察发现，整体上当选取样本数较少时，模型的准确性下降，PCA 消除了特征中有用的信息。当选取的样本数较多时，模型效果有所下降，回到未经降维处理的数据集水平。