计算机视觉和模式识别 作业 6

13331231
孙圣
计应 2 班

一、使用说明

通过 python3 adaboost.py 和 python3 svm.py 直接运行即可（需要安装 python3,numpy,scipy,scikit-learn）

测试环境：MAC OSX 10.11

Processor 1.6 GHz Intel Core i5

Memory 2 GB 1333 MHz DDR3

二、实验过程

一开始尝试使用 C++和 opencv 库进行训练和学习。

首先要做的就是从原始文件中读取相应的像素和标记信息。

根据网站上提供的数据存储格式[1]：

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel

先定义两个 header 结构体，保存从原始文件中读到的值：

```
// The type must be unsigned, otherwise error may occur while reading number.
struct labelHeader {
    unsigned char magicNumber[4];
    unsigned char number[4];
};

// Image header, which is the first 16 bytes of the file.
struct imageHeader {
    unsigned char magicNumber[4];
    unsigned char number[4];
    unsigned char rows[4];
    unsigned char cols[4];
};
```

对于图像和标记的读取基本一样，因此这里以读取图像为例[2]：

首先打开文件流，利用 read()函数读取相应的 header 信息：

```
fs.read((char*)(&header), sizeof(header));
```

然后，利用定义的 charToInt()方法，将 unsigned char 转换为 int。这里是通过左移 8 位相加的方法实现。需要注意的是：一定要把信息定义成 unsigned char，否则在这一步是无法正确转换的。

之后进行判断 magic number 是否正确，如果不正确直接退出。

对于每个像素的读取同样是利用 fstream 提供的 read()函数：

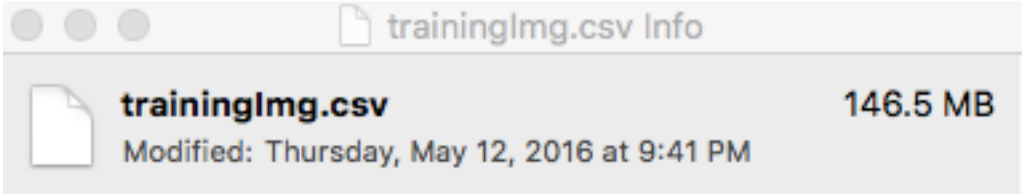
```
int totalSize = number * size;
unsigned char* tmp = new unsigned char[totalSize];
fs.read((char*)tmp, totalSize);

Mat ret(number, size, CV_8UC1, tmp);
```

然后直接利用 opencv 中 Mat 的构造函数转换成 Mat。

完成了数据的读取，接下来就是模型的训练了。查找了 opencv 的 adaboost 函数，发现其实并不是很容易使用。于是考虑将数据保存成 csv 文件[3]，然后用 python 读取进来进行训练。

尝试了一下之后发现并不是很现实，训练集的数据竟然有 140M 左右，如果用 python 读取会浪费很多时间进行 I/O：



因此考虑直接用 python 读取老师提供的.mat 文件[4]。虽然是 Matlab 保存的数据文件，但是 google 搜索后发现，其实读取.mat 的函数已经封装在 scipy 之中，只需要利用 loadmat()方法就可以将数据读取到 numpy 的 array 中：

```
# Read in .mat
# Ref: http://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html
trainingImgMat = sio.loadmat('MNIST/mnist_train.mat')
trainingLabelMat = sio.loadmat('MNIST/mnist_train_labels.mat')
```

此时读取出来的还不是 array，而是一个 dict，里面保存了各种信息，例如版本号等等，因此要通过该数据集的名字作为下标来得到真正的 array：

```
trainingImg = trainingImgMat["mnist_train"]
trainingLabel = trainingLabelMat["mnist_train_labels"]
```

之后获得相应的维度信息，例如训练集样本数，feature 个数，测试集样本数等等。

然后我使用了 scikit-learn 库中的函数进行训练。

1. Adaboost：

一开始定义一个 AdaBoostClassifier，其中的 base_estimator 为决策树，同时评价标准默认为 Gini[5]。以上数据都可以相对应的改变，例如：可以把评价标准改成我们熟悉的熵的信息增益，也可以把 base_estimator 改成随机森林等等。

之后就只需要调用 fit()和 predict()方法便能训练和预测测试样本了。这个库还提供了 score 属性，用来衡量预测的准确性。同时，我又对错误预测的样本进行统计，输出相应的错误率：

```
# http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
bdt = AdaBoostClassifier(DecisionTreeClassifier(criterion="entropy"))
#bdt = AdaBoostClassifier(base_estimator=RandomForestClassifier())
bdt.fit(trainingImg, trainingLabel.ravel())

predict = bdt.predict(testImg).reshape(numberOfTestData, 1)
score = bdt.score(testImg, testLabel)
#predictProba = bdt.predict_proba(testImg)
wrongCount = numberOfTestData - np.count_nonzero(predict == testLabel)
```

由于一开始得到的错误率较大（约 27%），因此怀疑库函数提供的并不是 one-vs-all 的多分类方法，于是自己尝试实现了 one-vs-all：

```
trainingFullLabel = np.empty([10, numberOfTrainingData], dtype=np.int8)
predictProbaFull = np.empty([numberOfTestData, 10])

for i in range(10):
    trainingFullLabel[i] = (trainingLabel.transpose() == i)

for i in range(10):
    bdt = AdaBoostClassifier(DecisionTreeClassifier(criterion="entropy"))
    bdt.fit(trainingImg, trainingFullLabel[i].ravel())

    predictProbaFull[:,i] = bdt.predict_proba(testImg)[:,1]

predict = np.argmax(predictProbaFull, axis=1).reshape(numberOfTestData, 1)
```

先是要将一列的 label 扩展为 10 列，每一列对应一个数字，相应的列中元素的值为 0 和 1，分别代表是该数字和不是。

然后训练出 10 个 adaboost 分类器，并获得对每一个测试样本的正向预测概率。最后通过 numpy 提供的 argmax 函数，对每一行找到概率的最大值，返回相应的 index。该 index 就是我们最终的预测值。

测试完毕之后发现正确率依然没有改善，这说明库函数实际上是采用 one-vs-all 的方法解决多分类问题。正确率低的问题应该是其他参数没有调好所致（解决方案见实验结果部分）。

2. SVM

```
# SVM Ref: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
#clf = SVC()
# LinearSVM Ref: http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
clf = LinearSVC(dual=False)
clf.fit(trainingImg, trainingLabel.ravel())
```

SVM 的代码和 adaboost 基本一致[6]。唯一要注意的是，最好使用 linearSVM。一开始使用了普通的 SVM 进行训练，训练了 4 个小时依然没有结束。原因是 SVM 的复杂度较高，需要大量的计算，因此使用 linearSVM 可以减少训练时间，同时准确率又不受太大影响。初始化时根据官网的建议，将 dual 设置为 False，因为样本的数目大于 feature 的数目。

三、实验结果

1. Adaboost

(1) 缺省情况：

错误率相当高：为 27%。

```
Error Rate: 0.270100
Score: 0.729900
MacBook-Air-3:Ex6 sunsheng$ time python3 adaboost.py

real    4m2.379s
user    3m38.423s
sys     0m6.642s
```

(2) Estimators 数目从 50 增长到 100：

错误率改变不大，说明这个调整作用不大，花费的时间还翻倍了。

```
Error Rate: 0.270400
Score: 0.729600
MacBook-Air-3:Ex6 sunsheng$ time python3 adaboost.py

real    9m32.121s
user    7m38.554s
sys     0m16.522s
```

(3) 尝试限制决策树的深度为 1，2，4：

错误率逐步降低，但时间开销大幅度增加：

adaboost maxDepth1 E.csv

```
1  Error Rate: 0.256100
2  Score: 0.743900
```

adaboost maxDepth2 E.csv

```
1  Error Rate: 0.189500
2  Score: 0.810500
```

adaboost maxDepth4 E.csv

```
1  Error Rate: 0.122200
2  Score: 0.877800
```

深度为 2 的时间：

```
MacBook-Air-3:Ex6 sunsheng$ time python3 adaboost.py

real    8m30.794s
user    7m42.595s
sys     0m9.250s
```

深度为 4 的时间：

```
MacBook-Air-3:Ex6 sunsheng$ time python3 adaboost.py

real    20m2.126s
user    17m41.741s
sys     0m19.935s
```

(4) 使用熵代替 Gini 作为评价标准：
错误率大幅度降低，大约为 11%，时间开销基本不变

```
Error Rate: 0.117100
Score: 0.882900
MacBook-Air-3:Ex6 sunsheng$ time python3 adaboost.py

real    2m21.164s
user    2m0.775s
sys     0m4.880s
```

(5) 之后再次尝试增加 estimators 的个数至 200 和 400：
错误率变化不大，说明 50 个 estimators 足够训练了。

adaboost Est200 Entropy.csv

```
1  Error Rate: 0.115700
2  Score: 0.884300
```

adaboost Est400 Entropy.csv

```
1  Error Rate: 0.117500
2  Score: 0.882500
```

(6) 尝试将学习率降低或提高，从 1 设置成 0.5 和 2：
错误率依然变化不大。

adaboost LR0.5.csv

```
1  Error Rate: 0.115800
2  Score: 0.884200
```

adaboost LR2.csv

```
1  Error Rate: 0.115700
2  Score: 0.884300
```

(7) 使用自己实现的 one-vs-all：
错误率变化不大，说明库函数也是使用 one-vs-all 方法，而且效率更高。

adaboost one vs all.csv

```
1  Error Rate: 0.130500
```

```
MacBook-Air-3:Ex6 sunsheng$ time python3 adaboost.py

real    8m17.392s
user    7m24.000s
sys     0m10.290s
```

(8) 使用随机森林取代决策树作为 base_estimator：
错误率大幅降低至 3%，原因可能是随机森林对样本和 feature 随机采样训练出多个模型，因此模型的推广性更强。

```
Error Rate: 0.035600
Score: 0.964400
MacBook-Air-3:Ex6 sunsheng$ time python3 adaboost.py

real    21m20.006s
user    17m6.161s
sys     0m45.044s
```

(9) 对初始数据进行归一化，即所有像素除以 255：
错误率基本不变，但训练时间减少大约一半。

```
MacBook-Air-3:Ex6 sunsheng$ time python3 adaboost.py

real    13m25.205s
user    11m55.034s
sys     0m35.284s
```

2. SVM
(1) 普通的 SVM 训练时间过长，4 小时内没有成功得到模型。

(2) 换用 LinearSVM：
训练时间缩短为一个半小时（可能比实际开销更长，因为在训练的时候同时有在训练神经网络），错误率约为 8%。

```
Error Rate: 0.082700
Score: 0.917300
MacBook-Air-3:Ex6 sunsheng$ time python3 svm.py
==Finish fitting==


real    88m33.839s
user    75m5.572s
sys     1m47.647s
```

(3) 对初始数据进行归一化，即所有像素除以 255：
训练正确率变化不大，但训练时间大幅度减少：

```
MacBook-Air-3:Ex6 sunsheng$ time python3 svm.py
==Finish fitting==


real    2m51.359s
user    2m31.982s
sys     0m7.745s
```

参考资料：

[1] MNIST 数据格式
http://yann.lecun.com/exdb/mnist/
[2] MNIST 数据读取
http://blog.csdn.net/sheng_ai/article/details/23267039
[3] opencv 保存 Mat 数据
http://stackoverflow.com/questions/16312904/how-to-write-a-float-mat-to-a-file-in-opencv
[4] scipy 读取.mat 文件
http://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html
[5] adaboost
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
[6] SVM
http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html