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 0SX 10.11

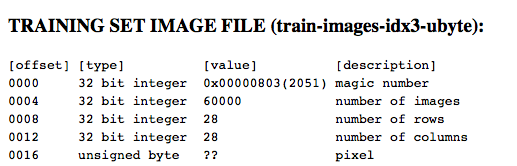


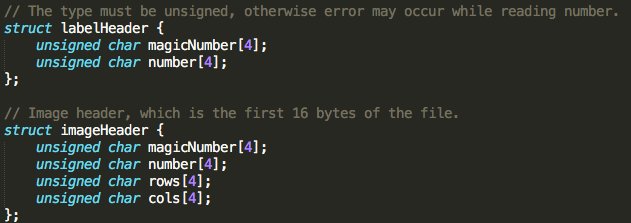
二、实验过程

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]：

先定义两个header结构体，保存从原始文件中读到的值：



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]：

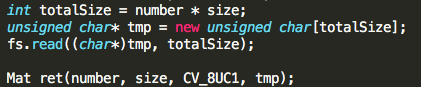
首先打开文件流，利用read()函数读取相应的header信息：



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。

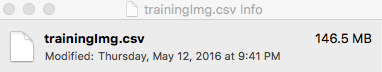
对于每个像素的读取同样是利用fstream提供的read()函数：



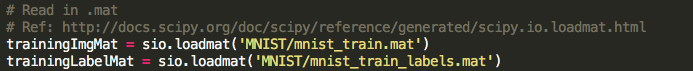
然后直接利用opencv中Mat的构造函数转换成Mat。

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：



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：



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。

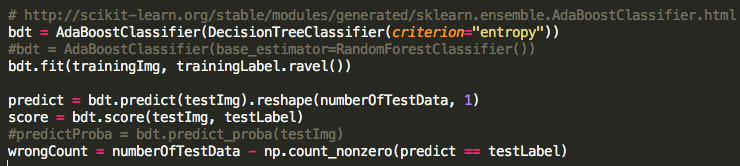
然后我使用了scikit-learn库中的函数进行训练。

1. Adaboost：

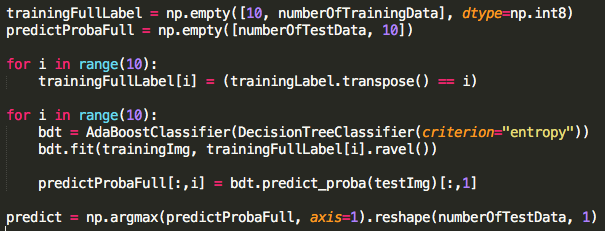
一开始定义一个AdaBoostClassifier，其中的base\_estimator为决策树，同时评价标准默认为Gini[5]。以上数据都可以相对应的改变，例如：可以把评价标准改成我们熟悉的熵的信息增益，也可以把base\_estimator改成随机森林等等。

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：



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：

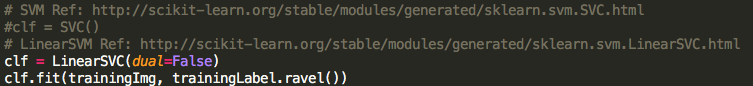


先是要将一列的label扩展为10列，每一列对应一个数字，相应的列中元素的值为0和1，分别代表是该数字和不是。

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。

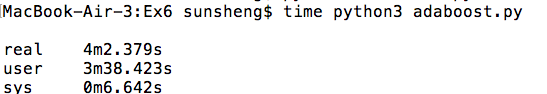
三、实验结果

1. Adaboost

(1) 缺省情况：

错误率相当高：为27%。

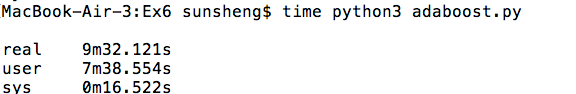




(2) Estimators数目从50增长到100：

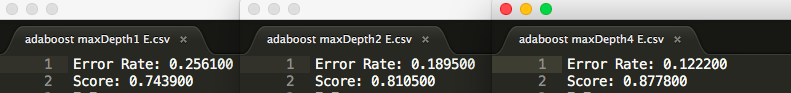
错误率改变不大，说明这个调整作用不大，花费的时间还翻倍了。



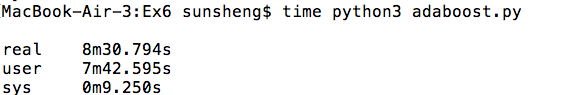


(3) 尝试限制决策树的深度为1，2，4：

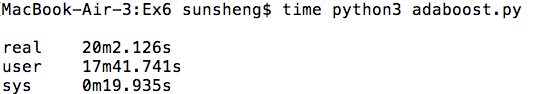
错误率逐步降低，但时间开销大幅度增加：



深度为2的时间：



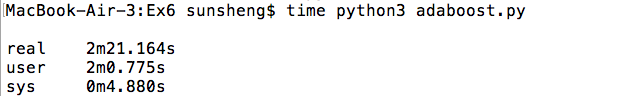
深度为4的时间：



(4) 使用熵代替Gini作为评价标准：

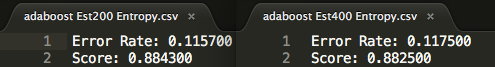
错误率大幅度降低，大约为11%，时间开销基本不变





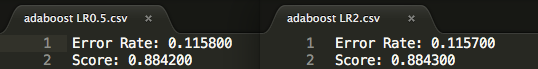
(5) 之后再次尝试增加estimators的个数至200和400：

错误率变化不大，说明50个estimators足够训练了。



(6) 尝试将学习率降低或提高，从1设置成0.5和2：

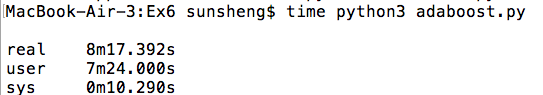
错误率依然变化不大。



(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。

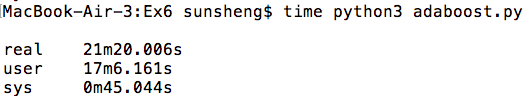




(8) 使用随机森林取代决策树作为base\_estimator：

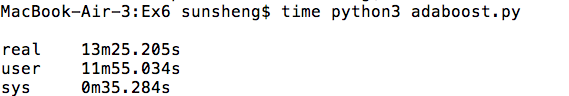
错误率大幅降低至3%，原因可能是随机森林对样本和feature随机采样训练出多个模型，因此模型的推广性更强。





(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。



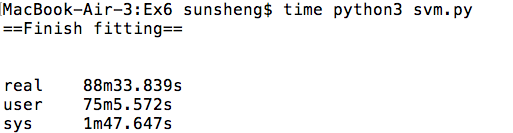
2. SVM

(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：

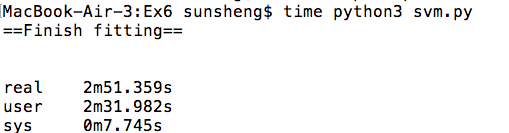
训练时间缩短为一个半小时（可能比实际开销更长，因为在训练的时候同时有在训练神经网络），错误率约为8%。





(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：



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>