AI homework-3: Classification 制作报告

物理系 基科31 蒋文韬 2013011717

2015 年 5 月 17 日

1 Tasks

1.1 Perceptron

Implement the train function in perceptron.py
Implement findHighWeightFeatures(self, label) in perceptron.py

1.2 MIRA

Implement trainAndTune in mira.py

1.3 Digit Feature Design

Add new binary features for the digit dataset in the EnhancedFeatureExtractorDigit function in dataClassifier.py

2 代码实现与运行结果分析

2.1 Perceptron

首先阅读dataClassifier.py与util.py中的代码，了解了util.py中的Counter类及其相应methods。通过阅读网站上的内容，并结合perceptron.py中的已有代码，可以分析得Counter类即为实现weight vector的容器。PerceptronClassifier类中的self.weights中的每一项，即为不同识别对象(即程序中的不同label)的weight vector，如在图像识别数字这一例中，self.weights的每一项即为数字0 ∼ 9对应的weight vector，而实现weight vector的Counter中的每一项则由图像的每一个像素点的坐标与对应的权值组成。

对程序的具体实现有所了解之后，我们要做的便很明确了。对训练数据中的每一项，对每一个label用self.weights中对应该label的weight vector与训练数据的vector作内积计算score，得分最大的label即为程序识别出的label。但识别结果并不一定正确，因此对识别有误的情况，我们将所识别出的label对应的weight vector减去该训练样本的feature vector，而将正确的label对应的weight vector加上该训练样本的feature vector。

实现上述算法的对应代码后，对不同的迭代次数，程序运行情况如下：

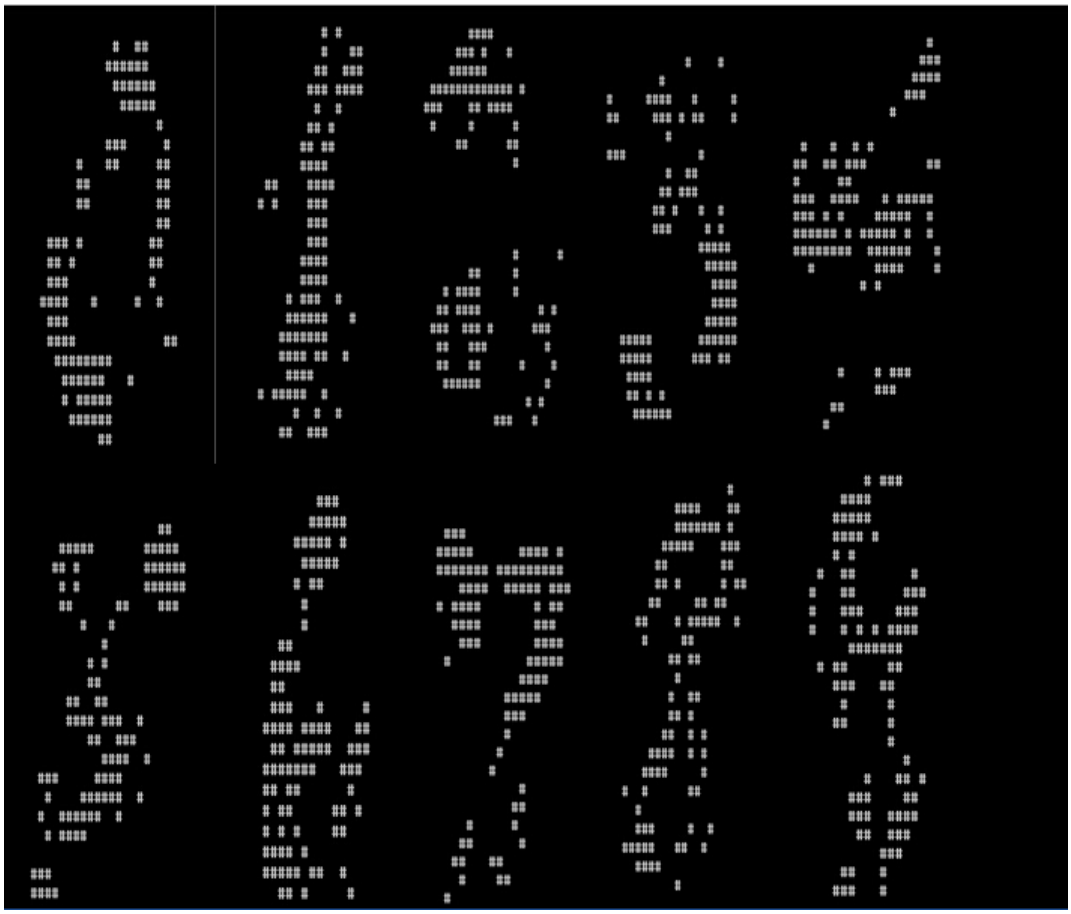
迭代次数	validation accuracy	test accuracy
1	63.0%	57.0%
2	60.0%	57.0%
3	55.0%	48.0%
4	55.0%	54.0%
5	56.0%	54.0%
6	56.0%	54.0%
7	56.0%	54.0%

可以看见，随着迭代次数增多，识别的成功率并不一定随之增加。训练的效果与训练样本以及测试样本息息相关。在第5次迭代后，测试的效果即已不再变化，表明weight vector已基本收敛。

2.2 Perceptron Analysis

由于Perceptron的可视化效果不好，因此对于图像识别数字的问题，我们可以通过选出weight vector中权值较大的feature(即像素点的坐标)打印出对应的图像，以使得学习的效果可视化。实现这一点，将self.weights中对应该label的weight vector拷贝出来，随后选出权值最大的feature，再将该feature的权值置负无穷，重复100次即可。

程序打印出的图像如下图(已进行拼接以方便查看)：



因此选择(a)选项。

2.3 MIRA

在前两问的基础之上，第三问较为简单。将perceptron.py中PerceptronClassifier类的train函数中的代码照写，在更新相应label的weight vector时对代码进行相应修改即可，即τ的计算按照下式

$$\tau = \min\left(C, \frac{(w^{y'} - w^{y})f + 1}{2||f||_2^2}\right)$$

进行，weight vector的更新按照下式

$$w^{y} = w^{y} + \tau f$$
$$w^{y'} = w^{y'} - \tau f$$

进行即可。需注意没有float型变量与Counter型变量直接相乘的运算，将Counter中的每一项单独乘以τ即可。还需注意对Cgrid中的每一个C都进行一次训练即可。

程序运行得validation accuracy为68.0%，test accuracy为57.0%。

2.4 Digit Feature Design

第四问是通过刻画其他的feature达到提高识别正确率的效果。我采用了题目中提到的想法，即对图像中hole的个数，或者说连通区域的个数进行计数。

首先实现getNeighbors(coord)函数，该函数返回当前坐标范围内的coord点的上下左右相邻点的坐标。利用visitedCoords记录所有访问过的点的坐标。findConnectRegion(coord)通过递归调用，达到访问到与coord相连的所有点，即访问该连通区域的效果。主函数中，变量numOfConnectRegion记录连通区域的个数。对未访问过的坐标使用findConnectRegion，并将numOfConnectRegion加一，即可达到计数连通区域个数的效果。最后，向features中增加'numOfConnectRegion'项，值为numOfConnectRegion即可。

但运行程序，发现识别的正确率并没有发生变化。猜想为数字图片中实际可能的连通区域数目情况也就为1、2等，情况较少，区分不够明显。于是将numOfConnectRegion = 1时的'numOfConnectRegion'项赋值为0，其余情况赋值为numOfConnectRegion。测试程序，validation accuracy与test accuracy均恰为84.0%，成功通过测试。

3 总结与感想

这次作业的完成比上一次感觉有所进步。通过前两次作业，我对python的基本使用已经较为熟练。

通过这次作业，我对机器学习有了更加具体与深入的了解与实践。对上课时有过讲授但在课堂的有限时间内无法详细讲清的程序实现的方法有了实际操作的经验。在通过应用机器学习进行分类上，既体会到了不同学习算法的区别，也在图像识别这一具体问题上进行了问题刻画方面的优化。另一方面，机器学习这一部分的内容极其丰富，这次作业考察到的只是一小部分，还有很多部分需要我更多的了解与学习。

在编程上，由于对python中不同数据类型的method了解不够丰富，程序中也一定还有不少需要提升的地方。如第3问与第4问在运行autograder.py时十分缓慢，效率低下。希望在这些方面能够逐渐有所提升。