**中山大学移动信息工程学院本科生实验报告**

**（2015年秋季学期）**

课程名称：Artificial Intelligence 任课教师：饶洋辉

|  |  |  |  |
| --- | --- | --- | --- |
| 年级 | 13级 | 专业（方向） | 移动信息工程 |
| 学号 | 13354485 | 姓名 | 朱琳 |
| 电话 | 13726231932 | Email | [280273861@qq.com](mailto:280273861@qq.com) |

**一 说明**

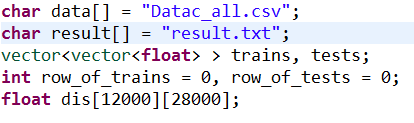
本次试验实现了6种算法——1NN，NB，ID3,C4.5,PLA,LR。是个整合型的报告。

**二 代码实现部分**

**(一)数据处理部分**

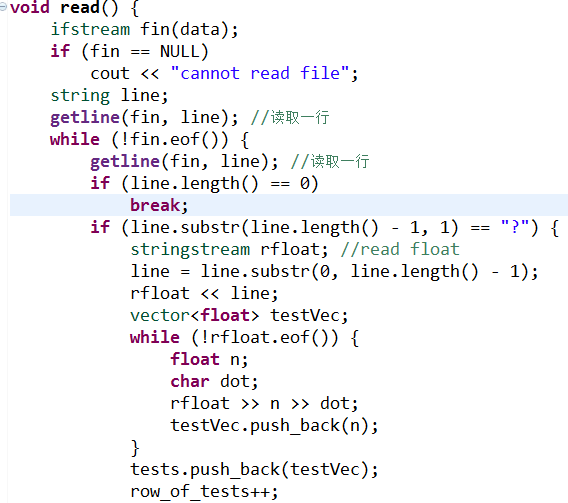
**【说明】**所有的实验代码均用以下数据结构，并且读取文件的函数没有基本没有发生改变。因此不再复述。

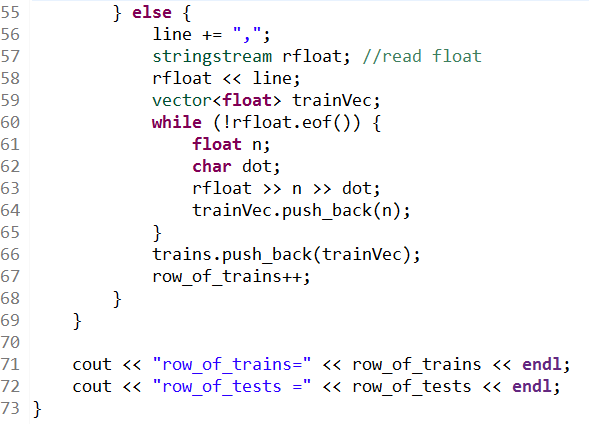
1. **数据结构**



Trains和tests的数据用一个vector<vector<float> >来表示，其中里面的vector用来盛放每一行的数据，外面的vector的size就是整个训练集或测试集的数量。

**2.读取文件**

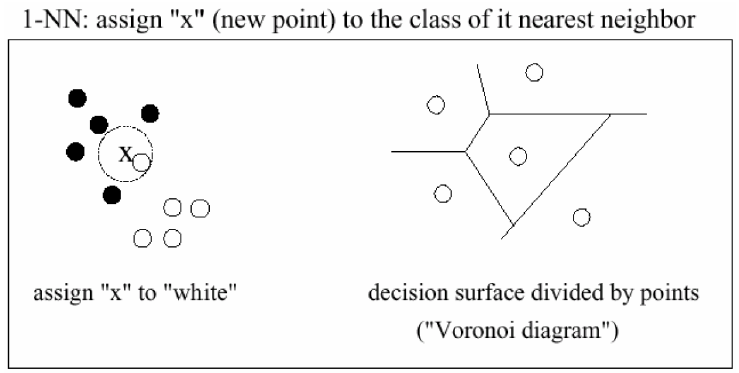




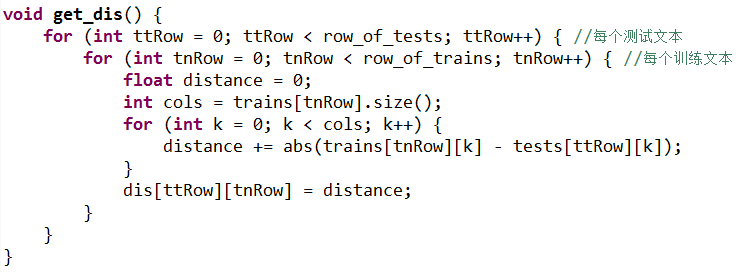
**(二)主要算法代码部分**

**【1.kNN】**

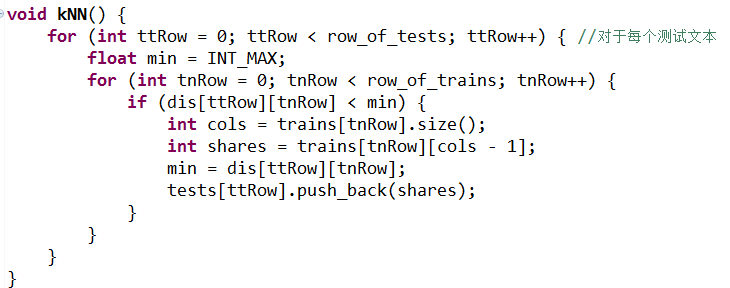
**【算法分析】**kNN，即k近邻算法，是将测试文本与训练文本的属性值作比较，选择距离最近的k个邻居，然后根据这些邻居最后的结果进行加权来预测测试实例的结果，距离可采用欧氏距离，曼哈顿距离，余弦距离等。关于kNN,之前做过两次实验，也比较熟悉了,大致流程就是读取文本->获取文本向量->计算各个测试文本对于各个训练文本之间的距离->选取最小（余弦距离选最大）那个距离->利用距离大小进行加权，距离越近，相似度越高，权重越大->通过权重和训练实例的结果值来预测测试实例的结果。



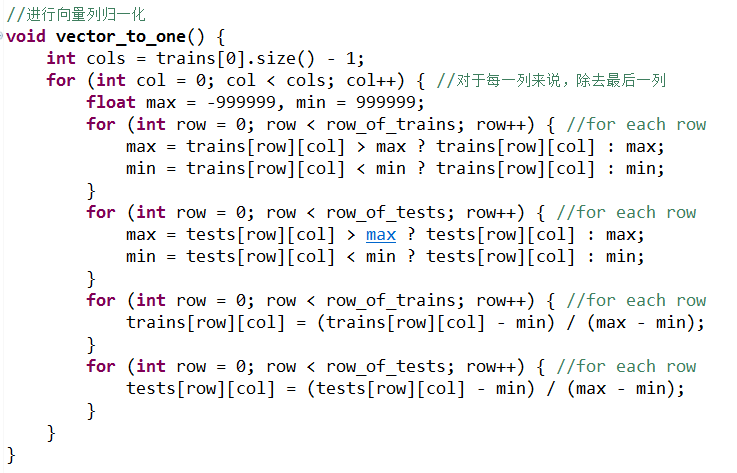
**(1)得到文本的距离——经过反复试验，采用曼哈顿距离作为最终的选择。**



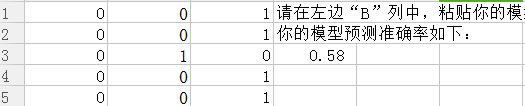
**(2)kNN具体计算部分**



**【kNN算法优化部分】**因为这次只要求实现1NN,因此我简化了原来的代码。开始我采用余弦距离并且没有归一化，得到的结果是0.56，后来对文本属性的每一列进行了[0,1]映射，即用归一化公式value= (value-min)/(max-min).通过遍历文本，得到某属性的最大值和最小值，再次循环遍历文本，使用上述公式进行归一化。具体代码如下：



归一之后采用曼哈段距离实现效果最好，为0.58



**【算法思考】**

感觉用交叉验证的方式确定N值的效果一定不错，但是这需要花费大量的时间来确定N值，这样会大大影响运行速度。

**【2.NB】**

**【算法分析】**

naive Bayesian，朴素贝叶斯算法。通俗来说，就是对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。因为lab5这个NB跟lab2的NB相差甚远，重新写了一遍朴素贝叶斯的实现，同时理了理思路。

naive Bayesian定义如下：

1、设x=\{a_1,a_2,...,a_m\}为一个待分类项，而每个a为x的一个特征属性。

      2、有类别集合C=\{y_1,y_2,...,y_n\}。

      3、计算P(y_1|x),P(y_2|x),...,P(y_n|x)。

      4、如果P(y_k|x)=max\{P(y_1|x),P(y_2|x),...,P(y_n|x)\}，则x \in y_k。

**现在的关键就是如何计算第3步中的各个条件概率。我们可以这么做：**

      ①找到一个**已知分类的待分类项集合**，即**训练样本集**。

      ②统计得到在**各类别下各个特征属性**的条件概率估计。即P(a_1|y_1),P(a_2|y_1),...,P(a_m|y_1);P(a_1|y_2),P(a_2|y_2),...,P(a_m|y_2);...;P(a_1|y_n),P(a_2|y_n),...,P(a_m|y_n)。

**PS：根据lab5数据集，只有两种分类，0or1；**

      ③如果各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：

P(y_i|x)=\frac{P(x|y_i)P(y_i)}{P(x)}

      因为分母对于所有类别为常数，因为我们只要将分子最大化即可，又因为各特征属性是条件独立的，所以有：

P(x|y_i)P(y_i)=P(a_1|y_i)P(a_2|y_i)...P(a_m|y_i)P(y_i)=P(y_i)\prod^m_{j=1}P(a_j|y_i)

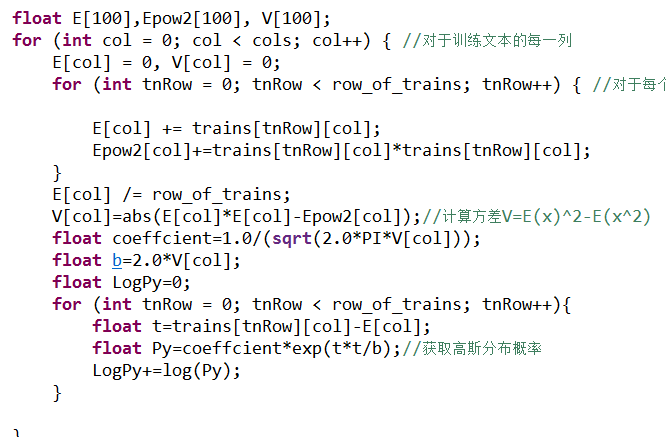
由于lab5的数据集是连续变量，因此我们使用高斯分布来确定

**总结上述模型的大致流程如下：**

确定特征属性->获取训练样本->对每个类别计算P(Yi)->对每个特征属性计算所有划分的条件概率->对每个类别计算P(Xi|Yi)\*P(Yi)->用P(Xi|Yi)\*P(Yi)最大项作为x所属类别。

 【绿色部分】是进行分类器训练，任务就是生成分类器，即计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计，并将结果记录。其**输入**是**特征属性**和**训练样本**，输出是分类器。【橘色部分】是应用阶段，任务是使用分类器对待分类项进行分类，其**输入**是**分类器**和**待分类项**，输出是待分类项与类别的映射关系。

**【代码优化部分】**因为需要使用到高斯分布，因此我们需要计算均值和方差，我采用公式V=E(x^2）-E(x)^2来计算，避免了频繁的平方和相加减的操作。同时，因为得到的条件概率可能很小，我采用了取对数的方式来防止数据下溢出。另外，为了防止出现=0的情况，采用拉普拉斯平滑来进行校准。



**【最终结果】**最终得到的结果为0.58。

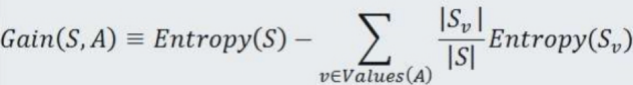
NB有一个缺点，NB模型假设属性之间相互独立，但是这个假设在实际情况中甚至在本次实验中并不总是成立的，这给NB模型的正确分类带来了一定影响。

**【3&4.决策树—ID3&C4.5】**

**【算法解析】**

D3算法的核心思想就是以信息增益度量属性选择，选择分裂后信息增益最大的属性进行分裂。

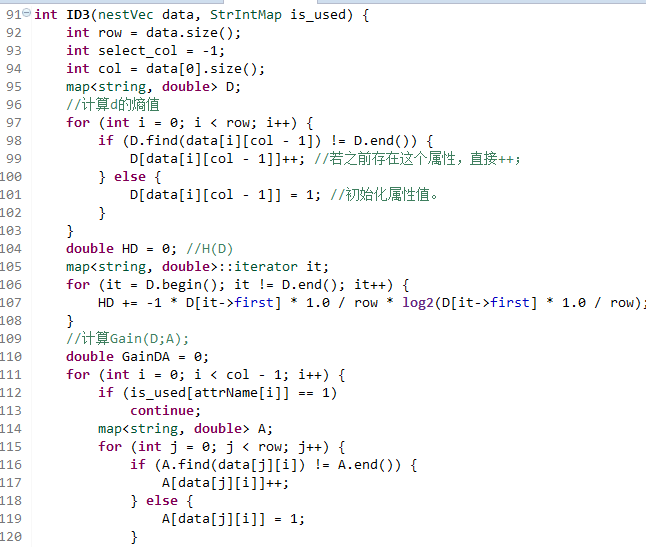
**增益是由信息熵进行计算的，具体公式如下：**

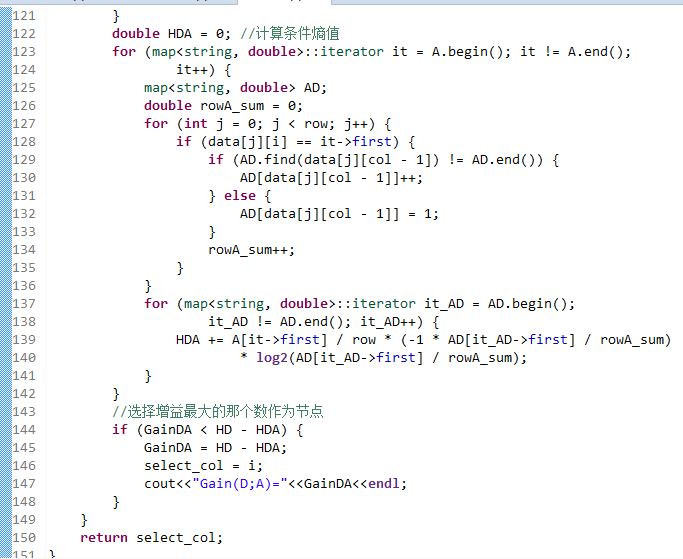


但是ID3算法存在一个问题，就是偏向于多值属性，比如按照key值进行分类，那么就会出现过拟合的现象。ID3的后继算法C4.5使用[增益率](http://en.wikipedia.org/wiki/Information_gain_ratio)（gain ratio）的信息增益扩充，试图克服这个偏倚。增益率用如下公式计算：split\_info_A(D)=-\sum ^v_{j=1}\frac{|D_j|}{|D|}log_2(\frac{|D_j|}{|D|}) gain\_ratio(A)=\frac{gain(A)}{split\_info(A)}

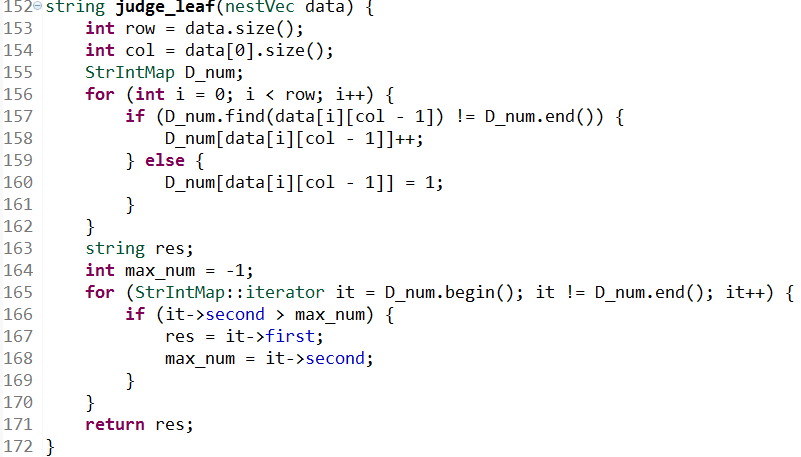
决策树首先需要算熵值，算增益（增益率），然后根据增益(率)选取节点，判断是否可以区分开所有训练集，如果可以就停止迭代，如果不行，则需要进行继续扩大树结构。必要时因为属性过多需要进行剪枝操作。

1. **计算熵值和增益，并选择最大增益的节点**

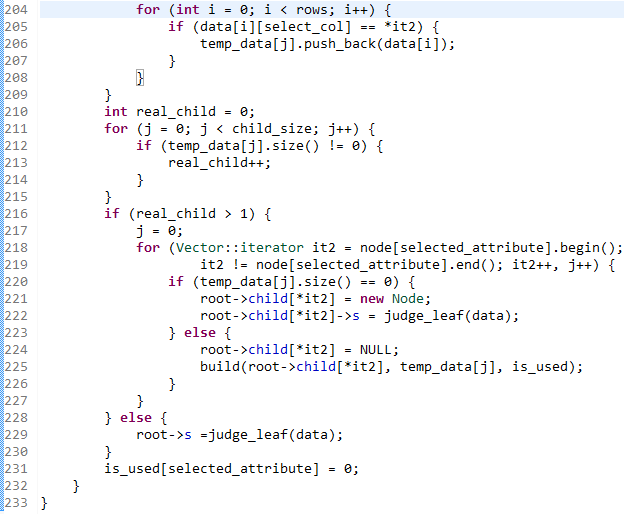
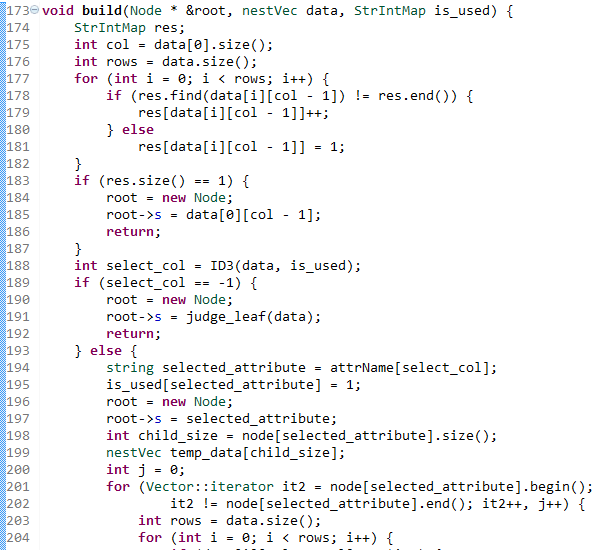




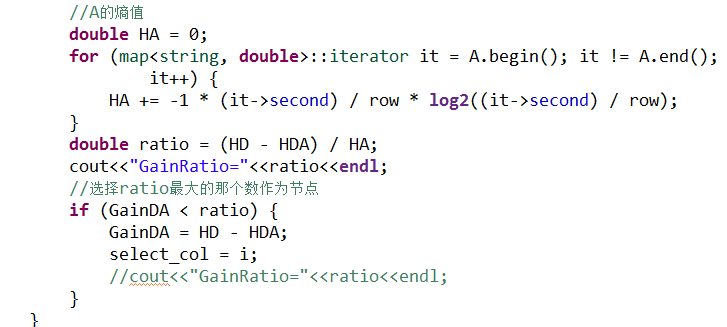
**(2)判断是否是叶子节点，即判断此树是否能够进行所有训练集的正确分类**



**(3)建树**



**(4)C4.5中增益率的计算—只是在ID3的基础上添加几行代码。**



**【算法分析】**最终得到的结果如下



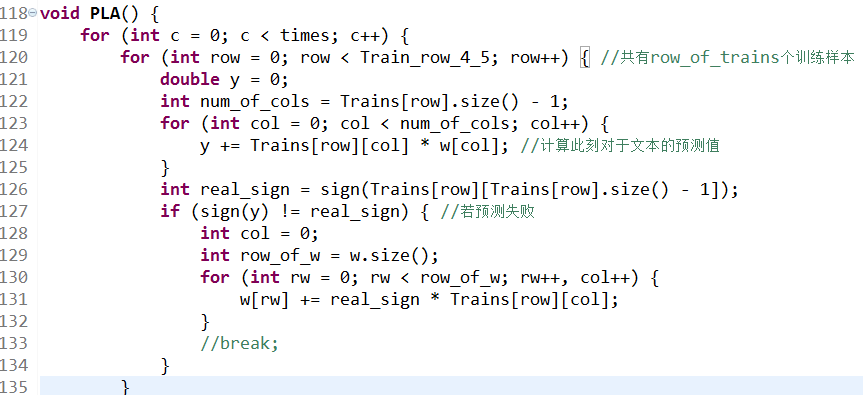
**【5.PLA+五折交叉验证】**

**【算法分析**】

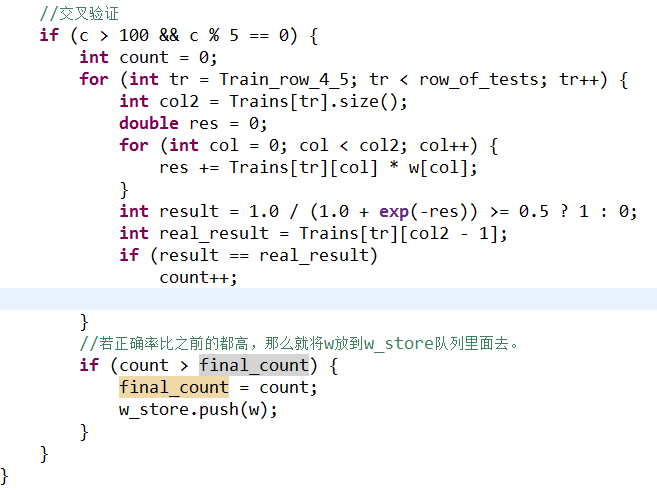
PLA，感知机学习算法，使用一个vector W来对所有的训练实例的X进行加权，加权之后得到的数的符号可用来判断结果，如果结果正确，则判断下一个训练实例，如果不正确则需要使用下列公式进行迭代:



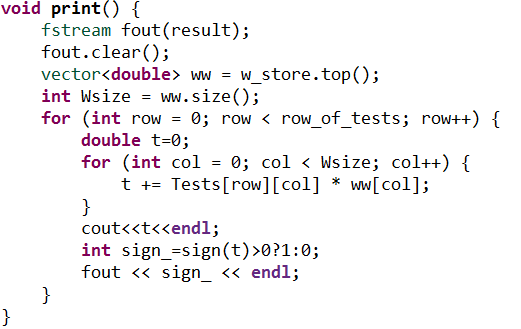
其中Yn有两种取值为1和-1。由符号函数来判断。得到更新的W之后再次进行迭代，在理想情况下迭代到可以正确区分所有的训练集为止。在大数据集下，我们几乎不可能有一个w满足所有的分类，因此上述的方式算法根本无法停止，因此我们需要选择合适的迭代次数，并且最好实现五折交叉验证来选取最好的W值。另外W的初值由自己决定，一般先确定训练实例的维度，再进行W的相应初始化，本次试验我将W的值初始化为全1。



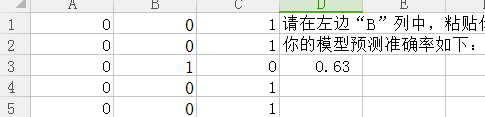
**【优化部分——交叉验证】**以训练集4/5作为训练文本，后1/5作为测试文本，验证得到效果最好的W值。



**【解释】**我是设定了一个栈w\_store()来专门存取进行交叉验证之后的w的值的。每迭代10次，就进行一次验证，如果此时验证的正确率大于原来的正确率，就将此事的w push到w\_store中去，在最后对测试文本进行验证的时候，再将w\_store中顶部的w拿出来作为最终的w即可。



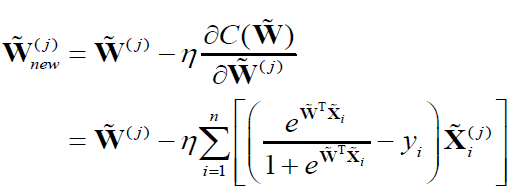
**【最终结果】**0.63



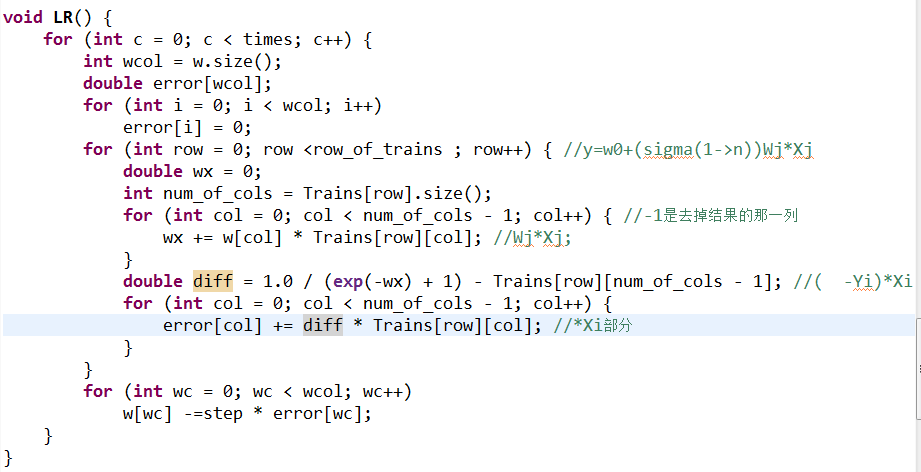
**【6.LR】**

**【算法解析】**

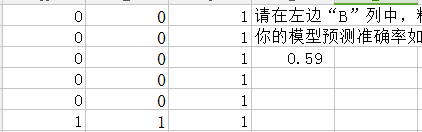
Logic Regression，逻辑回归算法，其实就是W**new**=W**old**-n\***▽**，其中**▽**是梯度，n 是步长。这个公式可以演化得到：



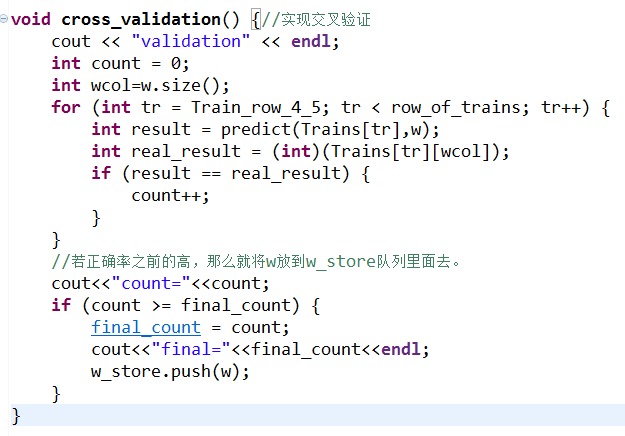
这本次实验中我先用一些常规的做法来实现。有实现进行向量列归一化，LR实现部分如下：



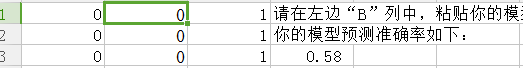
结果如下：



**【LR五折交叉验证部分**】



结果是：



我试着调了下训练集合的比例，并没有什么用，反而更小了。然后我将每次更新的结果输出出来，发现w迭代到一定的地步就可以预测成功所有的训练集（作为测试集）的文本。看来是过拟合了。

**二 实验结果比较**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **kNN** | **NB** | **决策树** | **PLA** | **LR** |
| **最好结果** | 0.58 | 0.58 | 0.61 | 0.63 | 0.59 |

综上，PLA最终的实现效果最好。