中山大学移动信息工程学院本科生实验报告

课程名称:*Artificial Intelligence*

年级	**1501**	专业（方向）	移动互联网
学号	**15352015**	姓名	曹广杰
电话	**13727022190**	**Email**	1553118845@qq.com

一、实验题目

逻辑回归

Logical regression for classification

二、实现内容

1. 使用梯度下降法实现逻辑回归的权值更新操作。
2. 将给出的数据划分为训练集与验证集。
3. 提交测试集的预测结果。
4. 对回归分类模型进行优化。

算法原理

逻辑回归基于PLA算法，弥补了PLA关于没有描述分类的概率的缺陷。将二者进行对比：

感知机算法：

- PLA划分的优化依据是实现的模型对训练集的符合度。

  即PLA总是以能够找到实现完美划分训练集的模型为终止目标。

- PLA认为任何一个元组的似然度没有意义。

  PLA并不考虑关于似然度的信息，对PLA来说，有一点像和非常像二者是没有差距的。

而对于逻辑回归算法而言：

- 逻辑回归的优化依据是实现的模型对训练集的最佳拟合。

  即逻辑回归不仅仅考虑当前的模型是否可以完成分类工作，还会考虑是否是最优的划分状态。

- 逻辑回归使用了logistic函数描述当前分类元组的确定性。

  逻辑回归会衡量有多大的把握预测正确了。可以评估“有点像”和“非常像”的区别。

逻辑回归在根本上与PLA还是没有太大的区别，都是使用线性的分类模型。

逻辑回归将$\vec{W}\vec{X}$作为预测的评估量，并将其通过logistic函数映射到[0, 1]的区间，将映射的结果作为预测的把握。

在实现更新的时候，我们

- 将权值的序列$\check{W}$作为一个自变量；
- 通过似然函数表示准确率；
- 将权值的变化拟合进衡量当前模型预测的确定程度的式子中：

$$L(W)=\sum_{i=1}^{n} y_i \check{W}^T \check{X}_i - log(1+e^{\check{W}^T \check{X}_i})$$

这样就获得了预测的确定程度与权值序列之间的函数关系。我们期望当前的预测可以更加准确，即对于预测的结果更有把握，就需要获得这个式子的最大值——这正是我们迭代更新的目的。

伪代码

```
            权值序列 w <- 全是1的序列
            repeat
              for 每一个元组 e 属于训练集 do
                for 每一个维度 w[j] 在 w 中 do
                  w[j] -= 步长
                           *(预测为正类的概率-实际的标签值)
                           *当前维度的特征值
                  记录预测正确的数目
                end for
              end for
            until 到了指定的迭代次数
```

关键代码截图

由于本次实验的基本框架依然是PLA的基本框架，唯一的不同便是在更新权值序列的过程中有所不同。以下代码表示更新权值序列的部分：

```
// jury 审判，用于判决当前的预测结果是否准确
bool Tuple::Jury(vector<double> &w_set, bool fix){
    vector<double> w = w_set;
    double multp=mulvec(w_set, xi);// 权值序列与特征序列的矢量乘积
    double Hx = forHx(w_set, xi);   // 使用logistic函数计算把握程度；

    int size = w_set.size();
    short sign = 0; // 标记是否预测准确
    if((Hx>0.5)&&(label==1)){
        sign = 1;
    }else if((Hx<0.5)&&(label==0)){
        sign = 1;
    }else   sign = 0;

    if(fix){// 有修改权值序列的权限
        for(int wpin=0; wpin<size; wpin++){
            double eta = (Hx-label)*xi[wpin]; // 步长
            w_set[wpin] -= eta*(Hx-sign)*xi[wpin];// 更新权值序列
        }
    }
    // 返回预测的序列以及返回值

}
```

创新点与优化

训练集与验证集的划分

本次实验可用的数据集中共有8000个元组，即8000行。在实现将以上元组划分为训练集和验证集的时候，笔者陷入了沉思——对于验证集和训练集来说，什么样的数据集可以最大程度上地展示模型的性能呢？

所谓划分数据集，就是摘取原有数据集的不同段落重新组合成训练集和验证集，至于如何摘取如何组合则需要根据原有的数据集的性质来考虑，即整个数据集的8000行元组之间是否有某种关系。

- 如果初始数据集8000行元组之间是随机的：

  那么无论我们如何划分，产生的数据集都是一致的。都会是随机的大的数据集的随机的一部分，并不会改变其原有的随机属性。最起码在验证当前的模型方面，由于二者的随机分布是均匀的，不会造成过于明显的影响。

- 如果初始数据集的8000行数据是依据某种模型生成的：

  即在不同的位置或者不同的段落，该数据集的数据是来自于不同的数学模型——可能前4000行是一个比较明显的数学模型，接下来是1000行的过渡，后3000行是另一个数学模型的数据产出。此时如何划分就称为需要仔细考虑的问题。

决定是否使用随机抽样划分数据集。

- 如果使用非随机抽样，即使用该数据集的前n行作为训练集，剩余的部分作为验证集，则笔者面对的工作很可能是通过一个已有的数学模型去**预测该数学模型的发展趋势**，即通过一个模型预测另一个模型。相当于仅仅知道前50行的数据，去预测第100行的数据，已知的数据与未知的数据相关度较弱。
- 如果使用随机抽样。则当前可能存在的两个数学模型对笔者而言就是一个数据较为庞大的整体数学模型，**是一个数学模型**。笔者需要做的只是根据当前数学模型的已有数据去**还原整个数学模型**。这相当于在知道了第99行的数据之后，预测第100行的数据，已知数据与未知数据的相关性较强。

由于使用逻辑回归分类在更新权值的增广矩阵的时候，是通过迭代实现**当前数学模型的更新**。其目的是优化权值序列的增广矩阵使其更加符合当前的数学模型。可见这种方法**并不适合实现从一个模型到另一个模型的预测**，更适合实现数据的拟合，因而对于当前的数据集，应该使用均匀抽样的方法划分数据集，才可以展示逻辑回归的迭代更新操作是否有效。

```
for(行数 l -> 总行数)
    if(((l+1)%3) == 0)
        // 写入valid.csv文件
    else
        // 写入train.csv文件
    endif
endfor
```

即每3行选择一条元组加入validation文件。

正则化优化

由上文分析得知：

- 当前的实现模式是根据已经给出的数据，尽量还原原有的数学模型，继而预测。
- 拟合的方式还是非常基础的多项式拟合。
  - 因为多项式拟合的时候，高阶项的系数越大就越容易过拟合。

- 又因为根据迭代公式计算高阶项的系数是一件非常复杂的事。

- 故而使用一个非常强硬的方法：**直接限定所有的系数都不要太大，以期望实现简化模型的作用。**

为此在设置迭代更新的衡量值的时候，在追求值不断减小的衡量值上添加一个由权值向量的模表示的惩罚值。

初始的由似然函数计算的衡量值：

$$likehood = -\sum_{i=1}^{N}\{y_i log(h(x_i)) + (1-y_i)log(1-h(x_i))\}$$

添加正则项：

$$likehood = -\sum_{i=1}^{N}\{y_i log(h(x_i)) + (1-y_i)log(1-h(x_i))\} + \lambda\sum_{j=1}^{M}W^2$$

添加的行为就使得为了追求似然函数的值较小就必须做出妥协，需要在<u>似然函数计算值的绝对值更小</u>和<u>权值的模</u>更小之间寻求一个平衡。

此时的更新公式也需要相应地做出变化：

$$\begin{aligned}\breve{W}^j_{new} &= \breve{W}^j - \eta\frac{\partial L(\breve{W})}{\partial \breve{W}^j} \\ &= \breve{W}^j - \eta\frac{\partial(\sum_{i=1}^{n}(y_i\breve{W}^T\breve{X}_i - log(1+e^{\breve{W}^T\breve{X}_i})) + \lambda|\breve{W}^j|^2)}{\partial \breve{W}^j} \\ &= \breve{W}^j - \eta\sum_{i=1}^{n}[(\frac{e^{\breve{W}^T\breve{X}_i}}{1+e^{\breve{W}^T\breve{X}_i}} - y_i)\breve{X}_i^j] - 2\times\eta\lambda\breve{W}^j\end{aligned}$$

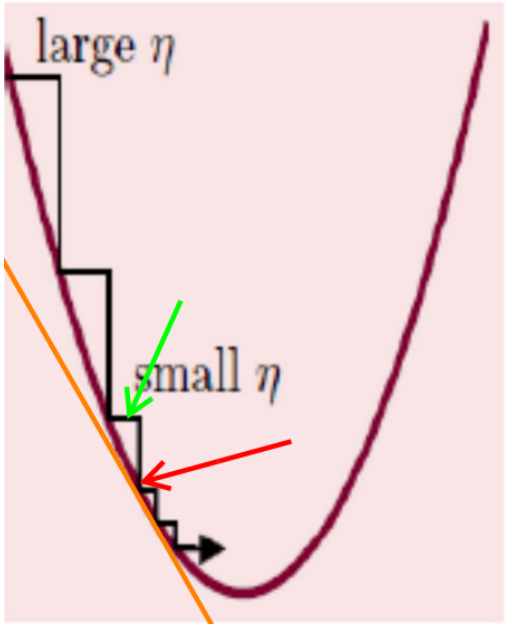
由于使用偏导数的时候，前两项都没有什么影响，无非就是平添了第3项，也是一个非常简单的式子。这里由于是使用公式编辑器，无法标注出来，就不做过多的解释了。

动态调整学习率

调整动态学习率之前，先梳理一下我们将要处理的函数：

$$L(W) = \sum_{i=1}^{n}y_i\breve{W}^T\breve{X}_i - log(1+e^{\breve{W}^T\breve{X}_i})$$

不出意外的话（在理论课上推导过一次了），该函数对于增广矩阵$\breve{W}$，即$\breve{W}$作为自变量的时候，该函数是类似于开口向下的二次函数的。而转化之后的迭代项$\frac{\partial L(\breve{W})}{\partial \breve{W}}$，这偏导数的形式则恰好表示该函数的斜率，即下降梯度：*这里偷偷盗一张图应该没人看见嘻嘻嘻*



分析该图：

- 红色箭头所指的点为迭代后的点，这里储存着新的权值序列的信息
- 橙色线表示该点的切线
- 绿色箭头所指的小平台则是步长η

从上一个点到当前红色箭头所指向的点，是通过一次迭代完成的。在那次迭代中，原有的点通过减去一个**步长和切线斜率的乘积**获得了当前红色箭头所指的点。

很熟悉的字眼吧？步长和该切线的乘积：

$$\eta\frac{\partial L(\check{W})}{\partial \check{W}}$$

不正是这个解析式吗。此外可以见到，斜率的变化是由快到慢的，也恰好符合我们对于步长变化的要求。为此，笔者直接将步长的值设置为斜率与一个常数的乘积，常数大小由实验调整而定。即可收获当迭代位置接近底部的时候，步长随之减少的效果。

```
        for(int wpin=0; wpin<size; wpin++){// 对权值序列的每一个值进行更新
            w_set[wpin] -= k// 常数
                           *abs( (Hx-label)*xi[wpin] )// 与 k 一起表示步长
                           *(Hx-label)*xi[wpin];// 表示斜率
        }
```

三、实验结果与分析

1. 实验结果示例展示

   由于分类的原则依然是根据所在的不同维度的权值进行划分，这里笔者就设计一个在维度层面上分明的小数据集作为训练集：

```
1,3,3,0,0,1
6,5,0,0,0,1
1,1,0,0,0,1
0,0,0,5,5,0
0,0,1,1,1,0
```

可以看到前3个属性促进该特征集划分为1，后两个属性的值若非零则促进其被划分为0；我们得到结果：

```
ans=1
ans=0


[main]:Calculating done.
[main]:shoot=2.  all=2.
[main]:Radio:100.000000%
```

2. 评测指标展示及分析

在笔者实现优化之前，使用LR算法，设置步长为1的初始状态，准确率为：

```
[main]:Analyzing validation set...
[Unused]/valid:num of attributes=40
[Unused]/valid:samplelines=2666
[main]:Analyzing done.
[main]:Calculating shoot count...
[main]:Calculating done.
[main]:shoot=1500.      all=2666.
[main]:Radio:56.264066%
```

继而使用动态调整学习率对步长进行优化，同时使用加倍的系数加快迭代的收敛速度（其实不加快也可以，只是要增加迭代次数），得到准确率为：

```
[main]:Analyzing validation set...
[Unused]/valid:num of attributes=40
[Unused]/valid:samplelines=2666
[main]:Analyzing done.
[main]:Calculating shoot count...
[main]:Calculating done.
[main]:shoot=1714.      all=2666.
[main]:Radio:64.291073%
```

由于步长如果直接使用斜率的值会过小，导致收敛速度过慢，所以在原有的斜率基础上，乘以一个较大的常数值，以增加其收敛速度，省去一些迭代次数。

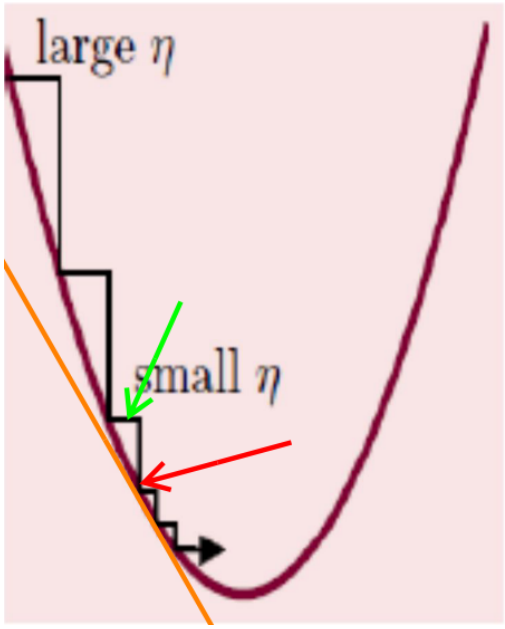
四、思考

- 如果把梯度为 0 作为算法停止的条件，可能存在怎样的弊端？

由于师兄事先提示我们，该问题的答案就在这一页。所以笔者仔细地阅读了这一页，得到信息：“直至梯度为 0 或者迭代足够多次”，那么答案就呼之欲出了，就是因为常规的迭代实在是难于收敛。如果实在要使用梯度为0作为算法终止的条件，就可能会导致迟迟不能终止，在迭代多次中逼近一个难于实现的值，这时候的迭代价值已经不大了。

- η 的大小会怎么影响梯度下降的结果？给出具体的解释，可视化的解释最好，比如图形展示等

  该问题笔者已经在之前的优化中分析过了。

  由于绿色箭头所指的小平台是步长η，在步长增大的时候，深度也就会随之增大，这使得收敛的速度加快。简单滴说，就是步长越大，收敛越快，反之亦然。

- 思考这批梯度下降和随机梯度下降的优缺点

  PPT上的图像比较抽象，笔者这里复述一下批梯度下降法与随机梯度下降法。

  批梯度下降：(谨慎地更新权值序列)，每次更新之前需要计算每一个样本的下降贡献，对所有的样本迭代一次并求和之后，将所得的和用于更新该权值序列。

  随机梯度下降：使用某一个样本的值代替整体的下降值，用于更新一次当前的权值序列，对权值序列的更新并不是每一个样本都作出贡献。

  批梯度下降：

  - 优势：下降的方向是明显的，由于权值序列的更新过程每一个样本都有参与，批梯度下降在复杂的模型中计算收敛的位置时，总是可以照顾到每一个样本，从而得到整体上的最优解。
  - 劣势：对于迭代权值序列的要求过高，在样本数量非常大的时候，每一次迭代更新都非常费时，因为要遍历每一个样本。

  随机梯度下降：

  - 优势：收敛速度快。因为在处理相似数据的时候，随机梯度下降省去了求和的过程，在计算复杂度上和收敛速度上都占尽了优势。
  - 劣势：对于复杂的模型，可能会拘泥于局部最优解。因为随机梯度下降在迭代的时候考虑的信息很不全面，尽管是下降，但是对于通盘的衡量并不准确，因而很容易局限于当前计算出的极小值而不是最小值。