中山大学数据科学与计算机学院

移动信息工程专业-人工智能

本科生实验报告

（2017-2018 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	**1515**	专业（方向）	**软件工程**
学号	**15352334**	姓名	**吴佳卫**

一、 实验题目

逻辑回归

二、 实验内容

1. 算法原理

- logistic 函数

理解这部分首先我回想起了 PLA 中的符号函数 sign()。这两个函数的作用都是一样的，因为在线性归回算法中我们的思路都是通过选择一个合适的向量 w，使我们可以根据w^Tx的值预测出 y 的值。在 PLA 中我们使用的思路是直接判断w^Tx的正负，也就是 y 是落在 w 的哪一侧。然而作为一个硬分类模型（直接由决策函数预测），PLA 自然有很多的不足之处。因此我们引入了 LR 算法，建立一个软分类模型，计算权重并预测目标的可能性。在 PLA 中我们通过符号函数将一个范围是(-∞，∞）的值映射到了{-1,1}上，而在 LR 算法中，我们通过 logisit 从函数，也就是：

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$$

这个函数将这个范围是(-∞，∞）的值映射到(-1,1)上。

- 最大似然估计算法

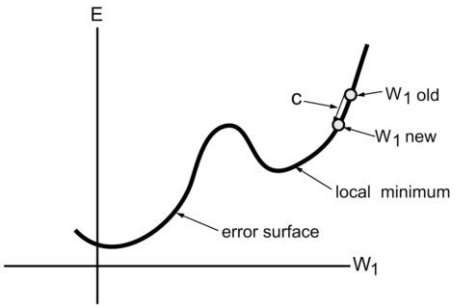
  在 PLA 中我们衡量模型是否已经收敛的办法是通过计算模型在验证机上的正确率。然而这次 LR 算法中我们通过计算似然函数来衡量，也就是：

  $$likelihood = \prod_{i=1}^{M} P(label|x_i) = \prod_{i=1}^{M} h(x_i)^{y_i}\big(1 - h(x_i)\big)^{1-y_i}$$

  似然函数的作用是，只要预测的结果$\mathrm{h}(x_i)$和实际的结果y_i相同，那么似然函数就会比较大。

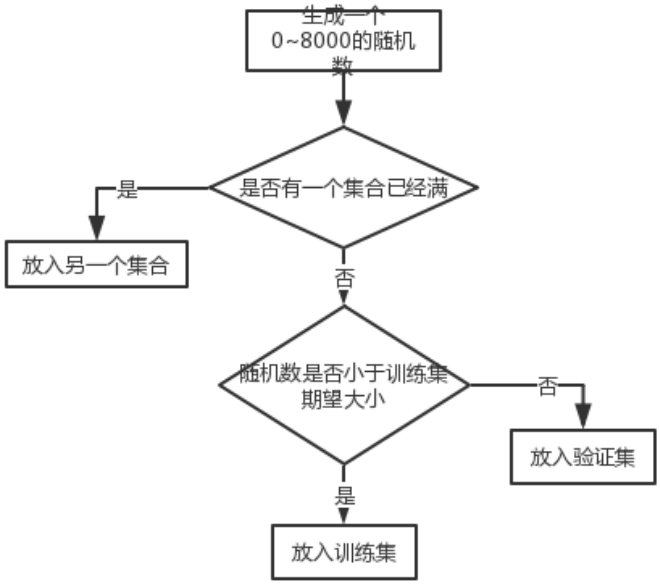
- 梯度下降法

  梯度下降法的思想在 PLA 中也使用过，基本的做法就是通过迭代使 w 达到最优值。

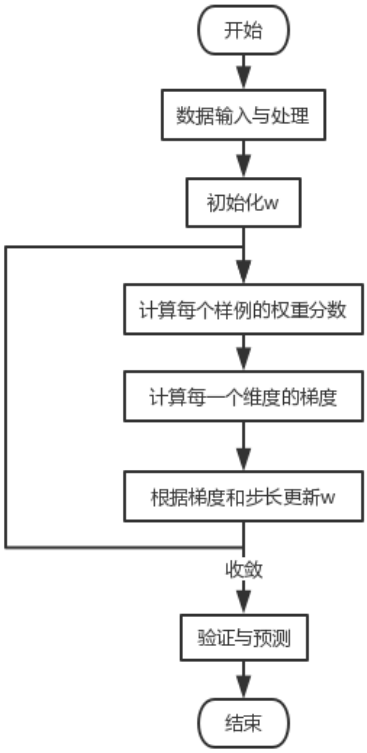


2. 伪代码

首先是划分训练集与验证集的原理，每次输入一个样本都通过调用一个函数随机决定这个样本的去向，如果某一个集合已经满了那么直接放入另一个集合。



至于其他部分的算法都可以放在同一个框架中，算法的核心就是根据流程不断更新 w 直到找到最优解。



3. 关键代码截图

第一部分是算法的核心部分，也就是通过逻辑回归算法迭代找到最优的 w。

```
w.resize(41);
for(int i = 0; i < w.size(); i ++) w[i] = 1;
norm(); //正则化
int tmp = 0;
for(int i = 0; i < iteration; i ++) //梯度下降
{
    vector<double> s; //计算每一个样例的权重分数
    s.resize(train_size);
    for(int n = 0; n < s.size(); n ++) s[n] = 0;
    for(int j = 0; j < train_size; j ++)
    {
        for(int k = 0; k < 41; k ++)
        {
            s[i] += w[k] * train_set[j].info[k];
        }
    }

    double grand = 0; //计算每一个维度的梯度
    for(int k = 0; k < train_size; k ++)
    {
        grand += (cal(s[k])-train_set[k].flag)*train_set[k].info[tmp];
    }

    w[tmp] -= step * grand; //更新一个维度的w
    tmp ++;
    if(tmp >= 41) tmp = 0;
}
```

还有是关于如何使用训练得到的 w 来对输入的验证集和训练集进行预测。由于计算的是当前向量属于特征“1”的可能性，因此最后判断的是这个数据是大于 0.5 还是小于 0.5 的。下面的代码是对验证集进行预测的代码。

```
double cnt = 0;
for(int i = 0; i < val_size; i ++)
{
    double tmp = 0;
    for(int j = 0; j < 41; j ++)
    {
        tmp += val_set[i].info[j] * w[j];
    }
    double result = cal(tmp);
    if(result > 0.5 && val_set[i].flag == 1) cnt ++;
    if(result < 0.5 && val_set[i].flag == 0) cnt ++;
}
cnt /= val_size;
cout<<'Accuarcy = '<<cnt<<endl;
```

4. 创新点&优化

本次实验主要使用了两项优化，优化对结果的影响写在第三部分。

第一项优化是队训练集的数据进行标准化（归一化），通过把一个范围不确定的数据集映射到（0,1）上，我们可以保证每一个特征对该数据的影响是相同的。使用的算法是

$$x^* = \frac{x - min}{max - min}$$

```
for(int i = 1; i < 41; i ++)
{
    double maxnum = 0;
    double minnum = 40;
    for(int j = 0; j < train_size; j ++)
    {
        if(train_set[j].info[i] > maxnum) maxnum = train_set[j].info[i];
        if(train_set[j].info[i] < minnum) minnum = train_set[j].info[i];
    }
    for(int j = 0; j < train_size; j ++)
    {
        train_set[j].info[i]=(train_set[j].info[i]-minnum)/(maxnum-minnum);
    }
}
```

第二项是随机的梯度下降。在没有使用该优化的算法中，每次我们都是按最大的幅度进行梯度下降，收敛速度很快但是执行的效率很低（因为要考虑所有的维度）。然而对于随机的梯度下降，我们使用任一维度的特征来代替整个向量，虽然瘦脸收敛的速度比较慢，但是每一次执行的步骤比较少，实际考费的时间也比较小。下图是批梯度下降算法。

```
for(int j = 0; j < 41; j ++)
{
    w[j] -= step * grand[j];
}
```

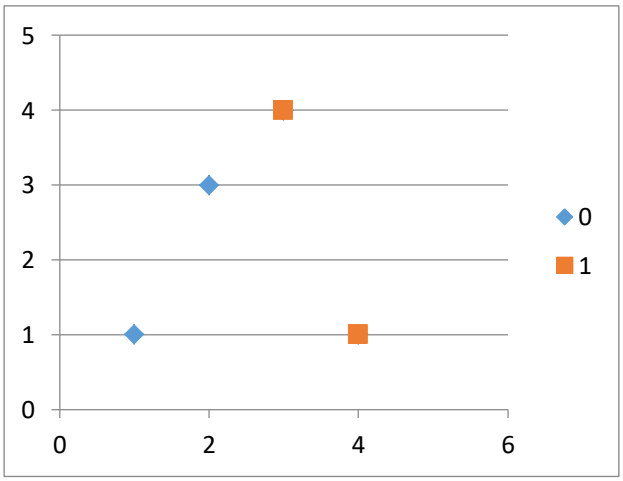
下图是随机梯度下降算法（其实并不是随机的，而是从第一个维度开始下降，一直到最后一个维度）

```
w[tmp] -= step * grand; //更新一个维度的w
tmp ++;
if(tmp >= 41) tmp = 0;
```

三、 实验结果及分析

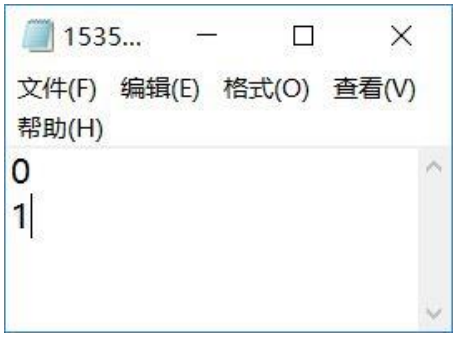
1. 实验结果展示示例

首先定义一个小数据集，维度是 2，一共有四个



对于测试集，定义两个点（1,3）与（5,2）

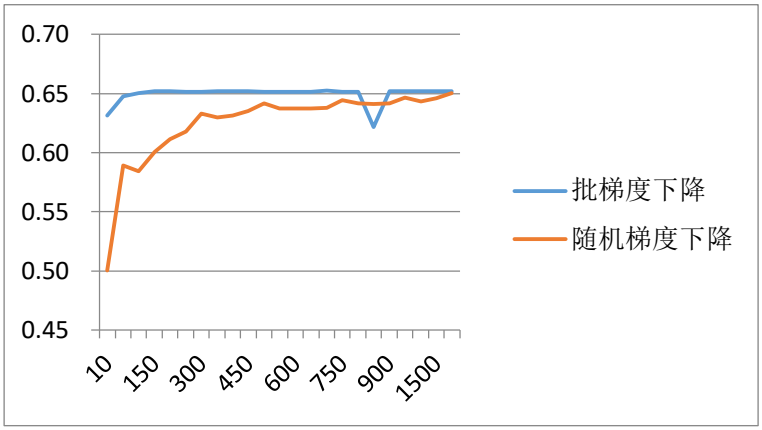
运行结果如下：



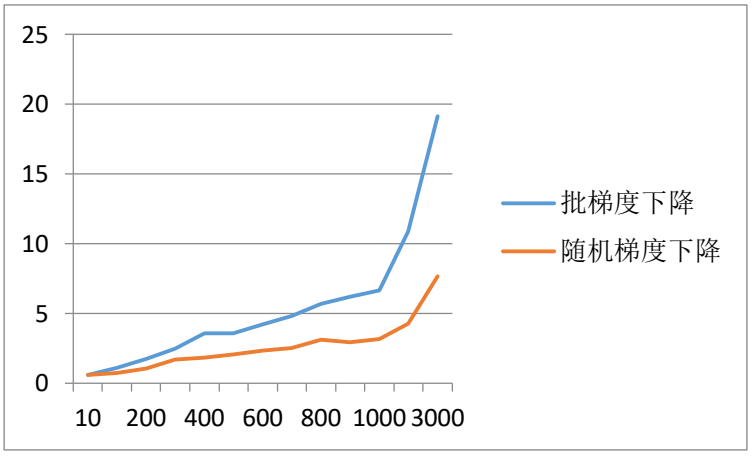
是与预期相符的。

2. 评测指标展示即分析

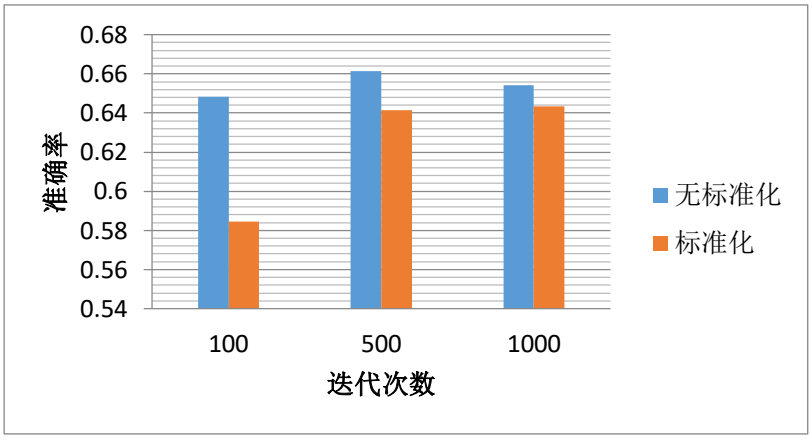
考虑两种不同的梯度下降方法：批梯度下降与随机梯度下降。首先从收敛的速度（多少次迭代后基本收敛）来说，批梯度下降的收敛是很快的，基本在 100 部以内就达到了最佳。



然而实际使用的时间，可以发现批梯度下降随着运行次数的增多，所需要的时间也明显的增加，而批梯度下降比较平稳。



至于是否标准化对准确率的影响，然而结果是标准化会降低准确率，迭代次数越少越明显。



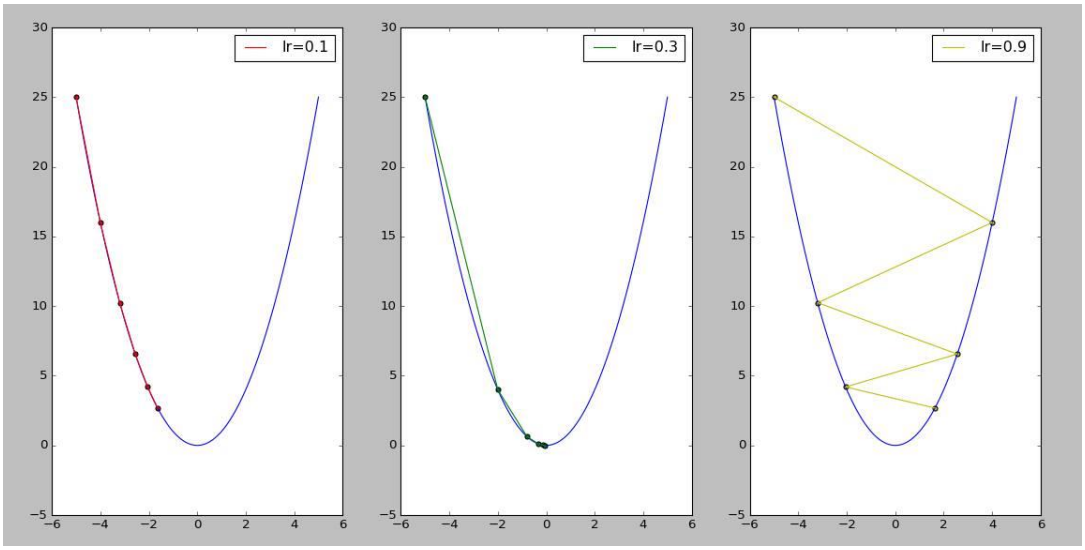
四、 思考题

1. 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？

因为数据集可能不是严格线性可分的，如果把梯度为0所谓停止条件可以导致算法一直无法停止。

2. 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

如图，过小的学习率会让梯度下降过慢，因为每次只从当前迈出很小的一步。过大的学习率会让梯度下降的过程中发生震荡，因为总是在下降的时候跳过了最优解，甚至可能导致无法收敛。



3. 思考这两种优化方法的优缺点

批梯度下降的优点是可以在很少的迭代次数内找到最优解 ,缺点是迭代次数增加以后需要花费大量的时间。

随机梯度下降的优点是花相对比较少的时间就可以收敛到最优解，缺点是不确定性比较高，需要迭代的次数可能比较多。