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		专业（方向）	**移动互联网**
学号		姓名	**Jw**

一、 实验题目

逻辑回归模型 -------- Logistic Regression Model

二、 实验内容

1. 算法原理

简介：逻辑回归模型是一个非线性的二分类模型，即特征向量的标签只有两种（1 或 0）。它是一种软分类模型，需要计算出属于每一种分类的概率。

决策函数：特征向量的每一个维度，都会对结果产生影响，所以与 PLA 一样，可以模拟一个带权重的分数：

$$s = \sum_{t=0}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

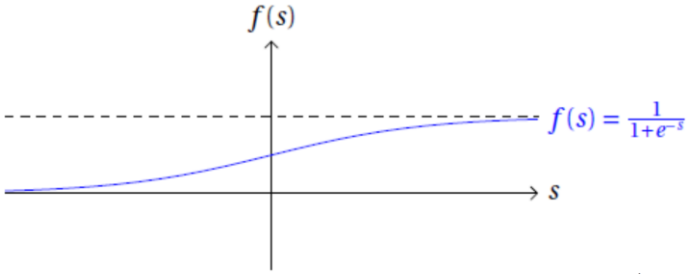
其中 $X_0 = 1$, $W_0 = \theta$（阈值）。表达式中的 X_i 表示第 i 维特征的值，W_i 表示第 i 维特征的权重($1<=i<=d$)，$W_i>0$ 表示该特征对正类别有正面影响，且值越大，正面影响越大，反之亦然。我们的目标，就是求出一个合适的权重向量 W, 用 W 去完成预测分类。

因此，跟 PLA 一样，我们需要一个决策函数来对 S 进行判决，然后根绝判决结果更新权重向量。逻辑回归在决策函数上与 PLA 的区别如下:

PLA	**逻辑回归**
$Yn = sign(w^T x_n)$	**$Yn = sigmoid(w^T x_n)$**

这里逻辑回归模型使用的决策函数是 sigmoid(s)，其表达式如下：

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



这个函数将带权分数从$(-\infty, \infty)$映射到了(0,1)，我们将映射后得到的结果作为数据属于正类

别的概率大小。观察该函数的图像，我们可以得到：

1. sigmoid$(-\infty)=0$，当加权分数无穷小，该数据属于正类别的概率为 0
2. sigmoid$(0)=0.5$，当加权分数为 0，该数据属于正/负类别的概率为 0.5
3. sigmoid$(+\infty)=1$，当加权分数无穷大，该数据属于正类别的概率为 1

由上可知训练的目标函数为：

$$\mathrm{h}(x_n)=\frac{1}{1+e^{-w^T x_n}}$$

预测得到的标签概率(后验概率)为:

$$\begin{cases} P(y_n=1|x_n)=\mathrm{h}(x_n) \\ P(y_n=0|x_n)=1-\mathrm{h}(x_n) \end{cases}$$

最大似然估计：在某种模型下利用给定数据 x 得到给定标签 y 的概率，是这个问题中的似然(likelihood)。对于某个数据样本，我们将其后验概率合并成如下形式:

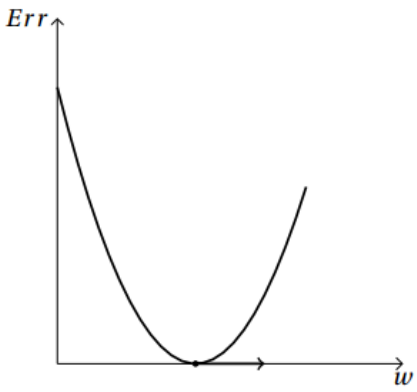
$$P(y_n|x_n)=\mathrm{h}(x_n)^{y_n}(1-\mathrm{h}(x_n))^{1-y_n}$$

则对整个数据集，似然函数是：

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

最大似然估计就是找到一组模型参数，似的似然函数最大，此时这组参数就是最优的。我们对似然函数求对数并取反，可以将最大化问题转为最小化问题，这个新的误差函数称为交叉熵，表达式如下(log 是以 e 为低)：

$$\begin{aligned}\min_{w}\quad \mathrm{Err(w)} &= -\log\textstyle\prod_{i=1}^{N}h(x_i)^{y_i}\big(1-h(x_i)\big)^{1-y_i} \\ &= -\sum_{n=1}^{N}y_n\log(\mathrm{h}(x_n))+(1-y_n)\log(1-\mathrm{h}(x_n))\end{aligned}$$



结合函数图像，Err(w)是一个连续可导，并且二阶可微的凸函数。根据凸优化理论，存在全局最优解，即∇Err(w)=0。我们不直接求解函数的零点，采用迭代最优化的方式去求解，由于∇Err(w)是凸函数，故我们只要沿着梯度下降的方向去更新求解 w，就一定能找到最优解，因为梯度是函数变化最快的方向。我们可以得到 W 的更新公式：

$$W_{t+1} = W_t - \eta \nabla Err(W_t)$$

其中，η>0 表示人共设置的梯度下降的步长(学习率)

交叉熵表达式的推导：

根据上面的描述，我们剩下唯一的问题，就是求出交叉熵的表达式，设 i 表示第 i 维度，n 表示第 n 个样本，推导如下：

$$\frac{\partial \mathrm{Err}(W^i)}{\partial W^i} = -\sum_{n=1}^{N} y_n \times \frac{1}{h(x_n)} \times \frac{\partial h(x_n)}{\partial W^i} + (1-y_n) \times \frac{1}{1-h(x_n)} \times -\frac{\partial h(x_n)}{\partial W^i}$$

$$= -\sum_{n=1}^{N} \left(\frac{y_n}{h(x_n)} - \frac{(1-y_n)}{1-h(x_n)} \right) \times \frac{\partial h(x_n)}{\partial W^i}$$

令 $u = -W^T x_n, v = 1 + e^u$，则

$$\frac{\partial h(x_n)}{\partial W^i} = \frac{\partial}{\partial W^i} \times \frac{1}{1+e^u} = \frac{\partial}{\partial W^i} \times \frac{1}{v}$$

$$= -\frac{1}{v^2} \times e^u \times \frac{\partial u}{\partial W^i} = -\frac{1}{v^2} \times e^u \times \frac{-\partial W^T x_n}{\partial W^i}$$

$$= \frac{1}{v^2} \times e^u \times \frac{\partial \sum_{k=1}^{d} W^k x_n^k}{\partial W^i}$$

观察式中最后一项可知只有当 k=i 时，最后一项结果非 0，所以我们可得：

$$\frac{\partial h(x_n)}{\partial W^i} = \frac{1}{1+e^{-W^T x_n}} \times \frac{1}{1+e^{-W^T x_n}} \times e^{-W^T x_n} \times x_n^i$$

$$= \left(\frac{1}{1+e^{-W^T x_n}} \right) \times \left(\frac{1+e^{-W^T x_n} - 1}{1+e^{-W^T x_n}} \right) \times x_n^i$$

$$= h(x_n) \times (1 - h(x_n)) \times x_n^i$$

将此结果代入$\frac{\partial \mathrm{Err}(W^i)}{\partial W^i}$可得：

$$\frac{\partial \mathrm{Err}(W^i)}{\partial W^i} = -\sum_{n=1}^{N} \left(\frac{y_n}{h(x_n)} - \frac{(1-y_n)}{1-h(x_n)} \right) \times h(x_n) \times \left(1 - h(x_n)\right) \times x_n^i$$

$$= -\sum_{n=1}^{N} \left(y_n \left(1 - h(x_n)\right) - (1-y_n) h(x_n) \right) \times x_n^i$$

$$= -\sum_{n=1}^{N} \left(y_n - h(x_n) \right) \times x_n^i$$

由此我们知道了 W 的更新公式：

$$W_{t+1} = W_t - \eta \nabla Err(W_t)$$

$$= W_t + \eta \times \sum_{n=1}^{N} \left(y_n - h(x_n) \right) \times x_n$$

然后我们只要不断迭代更新 W 直至它收敛或者达到指定迭代次数时停止即可。

2. 伪代码

训练部分：

Input： 特征向量集合{x}和标签集合{y}

Output： 最优解 W_{t+1}

Initialize： W_0 设为全 1 向量（也可以随机），设置最大迭代次数 maxStep，步长η

for $t = 0,1 \ldots maxStep$

  for $i = 0,1 \ldots d$

$$\nabla Err(W_t, i) = -\sum_{n=1}^{N}\left(y_n - \frac{1}{1+e^{-W^T x_n}}\right) \times x_n^i$$

  end for

  for $i = 0,1 \ldots d$

$$w_{t+1}^i = w_t^i - \eta \nabla Err(W_t, i)$$

  end for

  if $\nabla Err(\mathbf{W}) = 0$ *then*

    break

  end if

end for

预测部分：

Input： 测试特征向量集合{x}，训练好的权重向量 w

Output： 标签集合{y}

for $i = 0,1 \ldots N$

  $P_i = sigmod(dot(x_i, w)$

  if $P_i > 0.5$ *then*

    $y_i = 1$

  else

    $y_i = 0$

  end if

end for

3. 关键代码截图（带注释）

1.训练函数

我实现了 3 种写法，分别是梯度下降、随机梯度下降及随机梯度下降+动态步长。

```
57  def train(traindata, alpha, method="gradDescent", maxStep=100):
58      """
59      :param traindata: data matrix that used to train
60      :param alpha: learning rate、step length
61      :param maxStep: Max Iterations
62      :param method:  optimization method
63      :return: the result of prediction
64      """
```

```python
    print("training...")
    trainX, trainY = splitData(traindata) #得到特征向量矩阵X和标签Y
    W = np.ones((trainX.shape[1], 1)) #初始化权重向量
    N = trainX.shape[0]

    for step in range(maxStep):
        print(step)
        if method == "GD":
            # 梯度下降 grad Descent
            err = trainX.transpose() * (trainY - sigmoid(trainX * W))
            W = W + alpha * err
        elif method == "SGD":
            # 随机梯度下降 stochastic gradient descent
            for k in range(N):
                i = random.randint(0, N-1)
                err = trainX[i, :].transpose() * (trainY[i, 0] - sigmoid(trainX[i, :] * W))
                W = W + alpha * err
        elif method == "SSGD":
            # 随机梯度下降 + 减小步长
            for k in range(N):
                #变步长，考虑迭代次数和当前循环次数，分母+1防止除以0，最后+0.0001防止结果太小
                alpha = 4.0 / (1.0 + k + step) + 0.0001
                i = random.randint(0, N-1)
                err = trainX[i, :].transpose() * (trainY[i, 0] - sigmoid(trainX[i, :] * W))
                W = W + alpha * err
        else:
            print("Method Type wrong")
            exit()
    return W
```

2.预测函数、输入需要预测的矩阵以及训练好的 W，输出标签 Y

```python
def predict(dataMat, W, Type="test"):
    """
    :param dataMat: data matrix that are used to predict
    :Param W: Well trained weight vectors
    :Type:predict type, test or validate
    :return: the result of prediction
    """
    testX, testY = splitData(dataMat)
    n = testX.shape[0]
    result = []
    for i in range(n):
        label = 0
        #根据预测计算出的概率确定标签
        if sigmoid(testX[i, :] * W)[0, 0] > 0.5:
            label = 1
        result.append(label)

    if Type == "validate":
        # 给非测试集的数据计算准确率
        cnt = 0
        for i in range(n):
            if result[i] == int(testY[i, 0]):
                cnt += 1
        print("Acc = ", cnt/n)
        return result, cnt/n
    else:
        return result
```

4. 创新点&优化（如果有）

1. python 向量化运算，从一开始计算交叉熵和更新 W 我都是直接用的向量化运算

2.随机梯度下降。

由于当数据集比较大的时候，批梯度下降每一次更新要重新计算整个训练集的梯度，这样就会很慢，这里我们可以采用随机梯度下降去优化，即每一次只考虑一个样本的梯度，这样计算量会大大减小。但是这种方法需要设置足够的更新次数，次数太小的话由于每次更新是随机一个样本，训练度就不够，结果会很差。由于我的迭代次数设置为 90，这个次数用随机梯度下降是不够的，于是我在随机梯度下降的部分再增加一个循环，次数为训练集长度，这样我的随机次数就是 90*len(训练集)

```
76          elif method == "SGD":
77              # 随机梯度下降 stochastic gradient descent
78              for k in range(N):
79                  i = random.randint(0, N-1)
80                  err = trainX[i, :].transpose() * (trainY[i, 0] - sigmoid(trainX[i, :] * W))
81                  W = W + alpha * err
```

3.随机梯度下降 + 动态步长

我们知道当凸函数接近于极值点的时候，其导数的值会趋向于 0。那么也就是说我们应该将初始学习步长设置较大一点，随着梯度下降不断减小步长，以期望它在极值点附近的震荡幅度能最小化。我这里将外循环迭代次数以及内循环训练集长度都考虑在内(作为分母)，步长随两者增加而减小，为防止分母为 0 应该加上一个 1。当分母很大时，为防止结果太趋近 0 而使步长太小，应在最后加上一个常数项确保步长不会太小，具体见代码：

```
82          elif method == "SSGD":
83              # 随机梯度下降 + 减小步长
84              for k in range(N):
85                  #变步长，考虑迭代次数和当前循环次数，分母+1防止除以0，最后+0.0001防止结果太小
86                  alpha = 4.0 / (1.0 + k + step) + 0.0001
87                  i = random.randint(0, N-1)
88                  err = trainX[i, :].transpose() * (trainY[i, 0] - sigmoid(trainX[i, :] * W))
89                  W = W + alpha * err
```

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

一个简单的小数据集：

Id	Feature0	Feature1	Label
Train1	1	2	1
Train2	2	-1	0
Test1	3	3	?

手动计算，采用批梯度下降，步长设置为 1：

初始化W_0为{1,1}

计算每个样例的权重分数：

S1=1*1+1*1+2*1= 4

S2=1*1+2*1+(-1)*1= 2

每一维梯度计算：

$$\nabla \mathrm{Err}(W_0^0) = \left(\frac{1}{1+e^{-4}} - 1\right) * 1 + \left(\frac{1}{1+e^{-2}} - 0\right) * 1 = 0.86281087$$

$$\nabla \mathrm{Err}(W_0^1) = \left(\frac{1}{1+e^{-4}} - 1\right) * 1 + \left(\frac{1}{1+e^{-2}} - 0\right) * 2 = 1.74360795$$

$$\nabla \mathrm{Err}(W_0^2) = \left(\frac{1}{1+e^{-4}} - 1\right) * 2 + \left(\frac{1}{1+e^{-2}} - 0\right) * (-1) = -0.91676950$$

更新每一维的权重：

$$W_1^0 = W_0^0 - \nabla \mathrm{Err}(W_0^0) = 0.137189$$
$$W_1^1 = W_0^1 - \nabla \mathrm{Err}(W_0^1) = -0.74360795$$
$$W_1^2 = W_0^2 - \nabla \mathrm{Err}(W_0^2) = 1.91676950$$

迭代 1 次，结束学习，利用 W_1 对测试集进行预测：

$$\mathrm{P}(1|\mathrm{test1}, W_1) = \frac{1}{1+e^{-(1*W_1^0+3*W_1^1+3*W_1^2)}} = 0.97483156 > 0.5$$

所以预测标签为 1

代码跑梯度下降的结果：

```
问题    输出    调试控制台    终端
W = [[ 0.13718913]
 [-0.74360795]
 [ 1.9167695 ]]
P(label=1|test1) = 0.97483155802
label = 1
```

结果与我们手动计算结果完全一致

2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

我将 train.csv 的数据按照 4:1 划分成测试集和验证集，方法是随机划分，这里我记录下了其中一次的划分结果并且在迭代相同次数的前提下分别采用 3 种方法(批梯度下降、随机梯度下降、随机梯度下降+动态步长)去训练，然后用训练得到的权重向量去预测训练集和预测集并计算正确率。

批梯度下降：

```
问题    输出    调试控制台    终端
Method =  grad Descent
Max Step = 90
train data:
Acc =  0.6421666666666667
validate data:
Acc =  0.6675
```

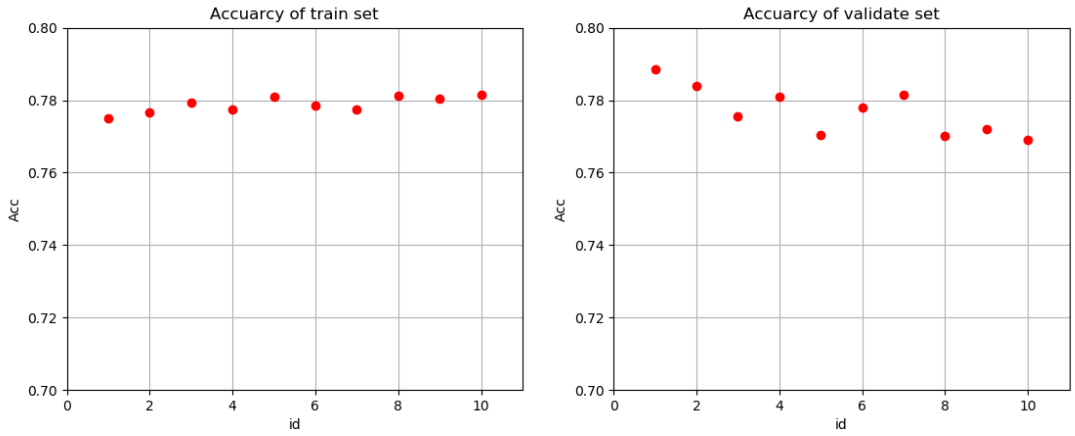
随机梯度下降：

```
问题    输出    调试控制台    终端
Method = stochastic grad Descent
Max Step = 90
train data:
Acc =  0.7768333333333334
validate data:
Acc =  0.771
```

随机梯度下降+动态步长：

```
问题    输出    调试控制台    终端
Method = stochastic grad Descent + dynamic step length
Max Step = 90
train data:
Acc =  0.7778333333333334
validate data:
Acc =  0.78
```

由图可知随机梯度下降+动态步长会比较好，然后选取该方法训练好的模型，再另外随机划分 10 次数据集，分别预测每次训练集和测试集得到的准确率的图像如下：



平均准确率为：

```
average accuarcy of train set
0.778833333333
average accuarcy of validate set
0.777
```

从图和平均准确率的结果可以看出，我用第三种方法训练出来的模型还算比较好，预测的准确率基本稳定在 0.77 以上。

四、 思考题

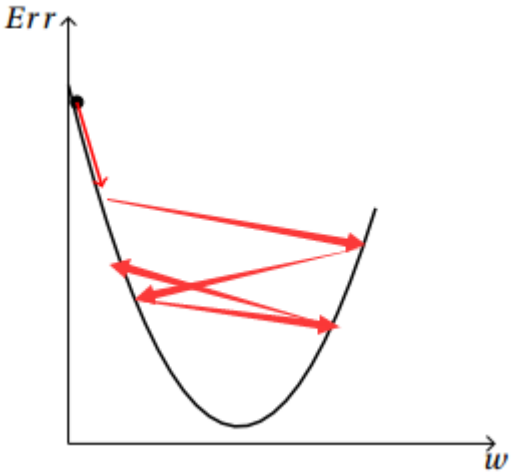
1. 如果把梯度作为 0 作为算法停止的条件，可能存在怎样的弊端？

答：1.普通的梯度下降法中步长是固定的，我们有可能在极值点附近一直来回震荡却始终无法达到极值点，这样算法就不会停止。

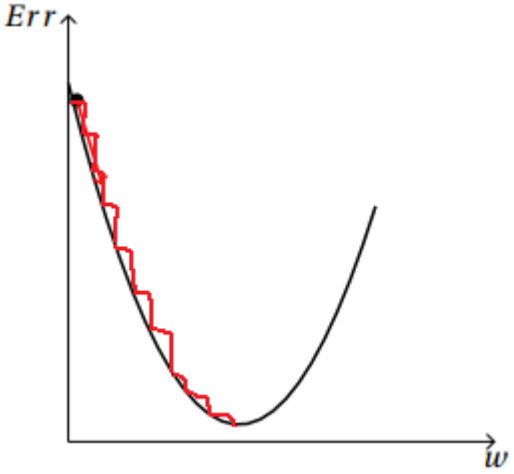
2.就算加入了变步长，在浮点数的计算中也非常难达到完全相等的条件，算法也不会停止。

2.学习步长 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？

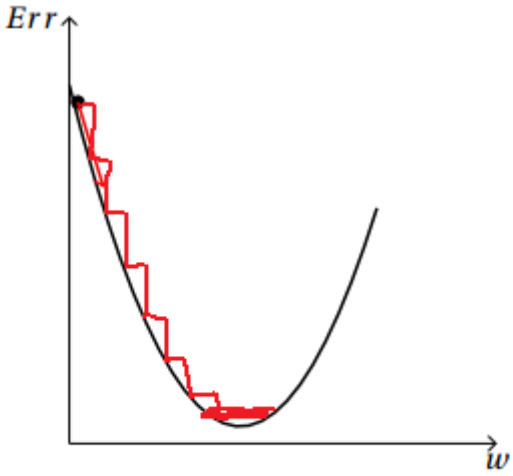
答：1. 若步长过大，则每次更新容易“变化过头” 造成剧烈震荡，始终无法收敛，如图:



2.若步长过小，这种情况能收敛，但是需要花费较长的时间才能收敛，如果迭代次数设置不够，有可能下降到一半就停止了。如图:



3.步长设置适中，能比较好的下降到极值点附近，并且在极值点附近较小的范围内震荡，可以近似看作收敛了。如图:



3. 批梯度下降和随机梯度下降的优缺点？

答：

批梯度下降：

优点：我们梯度下降的目标是希望权重向量 W 的每一维度都趋近极值点，而批梯度下降每次学习都使用了整个训练集，所以 W 的每次更新所有维度都会向着正确方向前进。

缺点：由于每一次更新都要重新计算整个训练集的梯度，计算复杂度高，耗时长，不适用于在线模型参数更新。

随机批梯度下降：

优点：每一次计算只要考虑一个样本的维度，大大提升了运算速度，可以用于在线模型参数更新。

缺点：需要设置足够大的随机次数，让其尽可能随机到每个训练集样本，否则会出现训练度不够，每次更新可能不会朝着正确的方向进行，这样训练出来的 W 对大多数样本的预测效果都不好。