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	1506	专业（方向）	移动信息工程（互联网）
学号	15352116	姓名	洪子淇
联系电话	13726205766	邮箱	1102229410@qq.com
开始日期	2017/11/20	完成日期	2017/11/22

一、 实验题目

1. 理解软分类模型以及硬分类模型；
2. 懂得逻辑回归的理论推导；
3. 实现逻辑回归算法并对此进行优化。

二、 实验内容

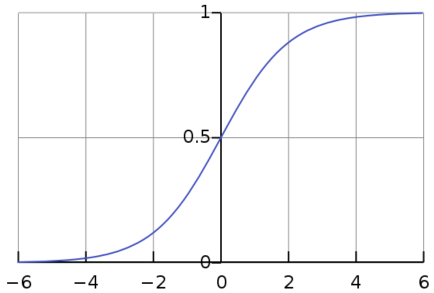
A. 算法原理

a) 逻辑回归

逻辑回归其实是一种分类方法，主要用于二分类。构造了一个预测函数 h：

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

利用了 Sigmoid 函数的特性，自变量大于 0 的时候收敛到 1，小于 0 的时候收敛到 0，换句话说就是自变量越大，是 1 的概率也就越大。如下图所示（图片资源：https://en.wikipedia.org/wiki/Sigmoid_function ）



根据 Sigmoid 函数，可以算出给定 x 向量，权重向量 w 下，label 值为 1 和为 0 的概率：

$$f(x) = \begin{cases} h(x) & if\ y = 1 \\ 1 - h(x) & if\ y = 0 \end{cases}$$

考虑整个文本集，根据贝叶斯法则可得到最大似然函数为：

$$\text{likelihood} = \prod_{i=1}^{M} h(x_i)^{y_i}(1-h(x_i))^{1-y_i}$$

根据最大似然估计，找到 likelihood 的最大值便表示对该 w 对样本划分最好，求 likelihood 的最大值，等价于对 likelihood 取对数求最大值，也等价于取对数再取负数求最小值，本实验采取最后一种方法，得到函数如下：

$$\text{Cost}(\,h(x),\text{y}) = -\sum_{i=1}^{M} y_i \log\big(h(x_i)\big) + (1-y_i)\log(1-h(x_i))$$

对 Cost 函数的w_j求偏导，即可得到w_j下的下降方向，通过设置w_j下降步长，更新每一个w_j，不断地迭代使整体 w 逼近最优解。

令 $\text{u} = 1 + e^{-w^T x}$，那么

$$\frac{\partial h(x_i)}{\partial \text{u}} = \frac{-1}{u^2},\quad \frac{\partial \text{u}}{\partial w_j} = -x_i e^{-w^T x}$$

$$\therefore \frac{\partial h(x_i)}{\partial \text{u}} \cdot \frac{\partial \text{u}}{\partial w_j} = \frac{x_i e^{-w^T x}}{(1+e^{-w^T x})^2} = x_i h(x_i)(1-h(x_i))$$

$$\frac{\partial Cost}{\partial w_j} = -\sum_{i=1}^{M}\left(y_i \cdot \frac{\partial \log\big(h(x_i)\big)}{\partial h(x_i)} - (1-y_i)\cdot\frac{\partial \log\big(1-h(x_i)\big)}{\partial h(x_i)}\right)\cdot\frac{\partial h(x_i)}{\partial \text{u}}\cdot\frac{\partial \text{u}}{\partial w_j}$$

$$= -\sum_{i=1}^{M}\left(y_i \cdot \frac{1}{h(x_i)} - (1-y_i)\cdot\frac{1}{1-h(x_i)}\right)\cdot x_i h(x_i)\big(1-h(x_i)\big)$$

$$= -\sum_{i=1}^{M}\Big(y_i \cdot \big(1-h(x_i)\big) - (1-y_i)\cdot h(x_i)\Big)\cdot x_i$$

$$= -\sum_{i=1}^{M}(y_i - h(x_i))\cdot x_i$$

最后得到权重向量的更新公式，其中α是更新步长：

$$w_j = w_j - \alpha\sum_{i=1}^{M}\big(h(x^i) - y^i\big)x_j^i$$

上面w_j的更新公式针对了整个样本，也称批量梯度下降更新。

b）随机梯度下降：

随机梯度下降算法和批量梯度算法只有一点不同，批量算法迭代更新 w 值的使用了整个样本值，算法复杂度为 O（Mn）。[注意：w 是同步更新的]。而随机梯度下降每读取一条样本就对 w 进行更新，算法复杂度为 O（n）。对于大数据，可能读取一部分数据就会使函数收敛，所有导致了可能不能收敛于最小值，而是在最小值附近震荡。下面是随机梯度下降的更新公式：

$$w_j = w_j - \alpha\big(h(x^i) - y^i\big)x_j^i$$

而无论是批梯度下降还是随机梯度下降，收敛条件的判断都是当这次的损失函数比上一次的损失函数要大，并且两次的权重向量的距离比小于一个极小数的时候判断为收敛。

$$\text{convergent}: Cost_{i+1} > Cost_i \ \&\& \ \frac{||W_{i+1} - W_i||_2}{||W_i||_2} < epsilon$$

c）向量化：

本次实验向量化主要用于矩阵相乘，提高效率，比如在批剃度下降w_j的更新公式可转换为矩阵运算，其中 M 为样本的个数，N 为特征值的数量：

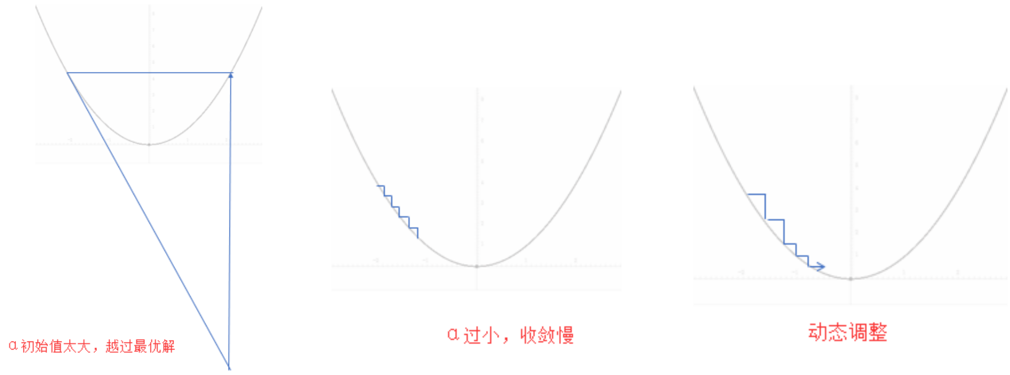
$$\mathrm{W}_{N,1} = \mathrm{W}_{N,1} - \mathrm{X}_{M,N}{}^{T}\ (\mathrm{h(X)}_{M,1} - Y_{M,1})$$

d）学习率：

即上文提到的学习步长α，一般通过手动调节α的大小得到迭代解。当α设置比较大的时候，一方面可以快速下降，不过可能会下降过多而导致越过最优解；当α设置较小的时候，虽然很难越过最优解，但同时也会导致收敛时间过长。所以在调节α需要选择较为合适的α才能逼近最优解。

动态学习率：字面理解就是动态调整学习率。当梯度下降降到最优解附近的时候，将学习率降低，避免越过最优解。在这次实验中，当越过最优解下一次步长减半，恢复上一次的权重向量 w 动态调整学习步长。

举个例子，对于凸函数$y = w^2$，通过梯度迭代求最优解，给定初值 w = -2，当学习步长$\alpha > 1$时就会越过最优解，设置太小迭代速度慢：



e）验证集：

本次验证集和上一次一样使用了 k-折交叉验证，将训练集随机分为 k 份，每份正负样例比例相同。用一份作为验证集，其余作为训练集，k 次循环验证。

B. 伪代码

a) 逻辑回归

Algorithm 1 Logistic Regression

```
function Sigmoid(X)
    return  1 / (1 + e^(-X))
```

function Logistic_cost(P, Y) # P is predicted label

    return $-\sum_{i=1}^{M} y_i \log(P) + (1-y_i)\log(1-P)$ #$y_i \in Y$

function Logistic_regression$(train_X, train_y, Weights, times)$

    if $Weights$ is empty: # dimensionality $train_X$: MxN $train_y$: Mx1

        $Weights = [0,0,0,\ldots,0]_{1XN}$

    end if

    $\alpha = 0.00001$

    $W_0 = Weights^T$

    $\mathbf{P} = \mathrm{Sigmoid}(train_X W_0)$

    $\mathrm{J}_0 = Logistic\_cost(\mathbf{P}, train_y)$

    for loop times do

        if is ***stochastic*** logistic regression

            $\#train_X{}^i$ means ith sample, $\mathbf{P}^i, train_y{}^i$ is analogous

            $W = W_0 - train_X{}^i(\mathbf{P}^i - train_y{}^i)$

        else if is ***batch*** logistic regression

            $W = W_0 - train_X{}^T(\mathbf{P} - train_y)$

        end if

        $\mathbf{P} = \mathrm{Sigmoid}(train_X W)$

        $\mathrm{J} = Logistic\_cost(\mathbf{P}, train_y)$

        if J is convergent

            Break

        else

            $\mathrm{J}_0 = \mathrm{J}, \quad W_0 = W$

        end if

    end for

    return W_0

C. 关键代码截图（代码+注释）

逻辑回归(批梯度下降):

```python
def sigmoid(X):
    return 1 / (1 + np.exp(-X))
def logistic_cost(p, train_y, w):
    # _p = p[:]
    _p = copy.deepcopy(p)
    _p[train_y == 0] = 1 - _p[train_y == 0]
    return -sum(np.log(_p)) / len(_p)
    # return np.dot((p - train_y).transpose(), (p - train_y)) / (2*len(p)) + lambda_ * np.dot(w.transpose(), w)
def logistic_regression(train_X, train_y, weights=None, times=10000):
    if weights == None:
        weights = [0.0] * train_X.shape[1] #initial value of weight vector
    alpha = 0.00001
    epsilon = 1e-10
    w0 = np.array([weights]).transpose() #40*1
    p = sigmoid(np.dot(train_X, w0)) # 7200*40 X 40*1 = 7200*1     1
    J0 = logistic_cost(p[:], train_y, w0)                                矩阵运算
    for t in range(times):
        p_y = p - train_y
        w = w0 - alpha * np.dot(train_X.transpose(), p_y) / len(p)     2
        # w = w0 - alpha * np.dot(train_X.transpose(), p_y) / len(p) - lambda_ * w0 / len(p) #40*1
        p = sigmoid(np.dot(train_X, w))
        J = logistic_cost(p, train_y, w)

        if J < J0: # not convergent
            J0 = J
            w0 = w
        else:
            # alpha /= 2     动态学习
            if np.linalg.norm(w - w0) < epsilon * np.linalg.norm(w0):
                print('times: ', t)
                break
    return w0 # 40*1
```

随机梯度下降：

```python
def stochastic_logistic_regression(train_X, train_y, weights=None, times=1000000):
    if weights == None:
        weights = [0.0] * train_X.shape[1] #initial value of weight vector
    alpha = 0.00001
    epsilon = 1e-20
    w0 = np.array([weights]).transpose() #40*1
    p = sigmoid(np.dot(train_X, w0)) # 7200*40 X 40*1 = 7200*1
    # print(p[:5])
    J0 = logistic_cost(p[:], train_y, w0)
    for t in range(times):
        i = random.randint(0, len(train_X) - 1)
        p_y = p - train_y
        train_Xi = train_X[i:i+1]   #1*40
        p_yi = p_y[i:i+1]           #1*1
        w = w0 - alpha * np.dot(train_Xi.transpose(), p_yi) # 40*1
        # w = w0 - alpha * np.dot(train_X.transpose(), p_y) / len(p) - lambda_ * w0 / len(p) #40*1
        p = sigmoid(np.dot(train_X, w))
        J = logistic_cost(p, train_y, w)
        if J < J0: # not convergent
            J0 = J
            w0 = w
        else:
            alpha /= 2
            if np.linalg.norm(w - w0) < epsilon * np.linalg.norm(w0):
                # print('times: ', t)
                break
    return w0 # 40*1
```

D. 创新点&优化

创新点：使用向量化运算，设置动态步长，实现了随机梯度下降。

三、 实验结果及分析

1. 实验结果展示示例

小数据集数据展示：

No	Attribute1	Attribute2	Label
train 1	1	2	1
train 2	2	-1	0
test 1	3	3	?

设置初始权重向量为$[\mathbf{1},\mathbf{1},\mathbf{1}]$,步长为 1，迭代次数为 1，验证模型：

——理论：

$$X1 = [\,1\;1\;2],\quad X2 = [1\;2-1]$$

$$h(x^1) = \frac{1}{1+e^{-(1*1+1*1+1*2)}} = 0.98201379$$

$$h(x^1) = \frac{1}{1+e^{-(1*1+1*2-1*1)}} = 0.88079708$$

$$w_1 = w_1 - \sum_{i=1}^{2}\left(h(x^i)-y^i\right)x_1^i = 1-(0.98201379-1+0.88079708) = \ 0.13718913$$

同理得到，$w_2 = \ -0.74360795$，$w_3 = 1.9167695$

那么预测值为：$P(1|test1,W) = \ \frac{1}{1+e^{-(1*w_1+3*w_2+3*w_3)}} = \ \ 0.97483156$

实验结果验证：

```
PS C:\资料\大三上\人工智能\实验\Lab5_LR> py .\LogicalRegression.py
[[ 0.13718913]
 [-0.74360795]
 [ 1.9167695 ]]
[[ 0.97488156]]
```

W

2. 评测指标展示分析

①批梯度下降的准确率与随机梯度下降的准确率以及时间的对比：

```
PS C:\资料\大三上\人工智能\实验\Lab5_LR> py .\LogicalRegression.py
K-Cross validation in logistic regression
Accuracy:
0.765
0.76125
0.79
0.78375
0.77625
0.7825
0.78375
0.77375
0.7775
0.78375
Time: 770.4003131389618 s
K-Cross validation in stochastic logistic regression
Accuracy:
.\LogicalRegression.py:33: RuntimeWarning: divide by zero encountered in log
  return -sum(np.log(_p))
0.67375
0.6775
0.65
0.725
0.67375
0.57875
0.65125
0.6625
0.6975
0.745
Time: 18.3135666847229 s
```

分析：根据实验结果分析，进行 10 次交叉验证，随机梯度下降的权重向量的更新只考虑一个随机样本，对比批梯度下降考虑全体样本，时间明显降低。不过由于个体难以概括总体损失程度，所以迭代比较曲折。

②固定步长与动态步长，准确率比较：

	准确率
初始步长0.001	0.505
初始步长0.0001	0.505
初始步长0.000001	0.779999
动态	0.77625

分析：当步长比较大的时候，由于不能收敛，所以会一直在震荡到循环次数为 0，又由于步长较大，没有逼近最优解，导致了准确率不高。当步长比较小的时候，迭代次数高，不断逼近最优解直至收敛，所以准确率会相对较高。

对于动态步长，设置初始步长没有很大关系，由于会动态降低，一旦损失比上一次高

就恢复上一次的权重向量，并且步长减半，使用动态步长可以较好地去逼近最优解，由于是动态变化，设置初始步长大，也会降低，最后会在靠近最优解的地方开始迭代，相比一开始固定降低的步长更具有弹性，时间上也会相对较少。

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?

梯度为 0 也就是说，下降到最优解。一般条件下，算法只能逼近最优解，很难准确地命中最优解，如果将梯度为 0 作为算法停止的条件，那么权重向量可能会一直在最优解附近震荡，而且一旦权重向量越过最优解，那么算法就会一直迭代而不能停止。

2.η的大小会怎么影响梯度下降的结果？给出具体的解释。

这个问题在算法原理上面提到，这里简单地重复一下。η如果设置太小，会导致迭代次数多，收敛速度慢，一般不会越过最优解，最后得到的权重向量比较逼近最优解。η如果设置太大，迭代次数减少，收敛速度快，不过也增强了越过最优解的概率。权衡速度与最优，η的值不能设置太大，也不能设置太小。也可以通过动态对η进行调整。

3.批梯度下降和随机梯度下降的优缺点。

这个问题在评测结果很容易看出来，随机梯度下降得到的结果比较随机，由于只是随机抽样对权重向量进行更新，具有局部性，以下是具体说明:

批梯度下降是根据整体样本的最大似然函数推导出来的，所以根据损失函数对权重向量进行更新，也是考虑了整体样本的损失程度，使用批梯度更新最后得到的权重向量对整体样本的预测损失较小，也就是可以较为准确对样本进行划分。不过批剃度下降有个很明显的缺点，如果样本数目较大，那么每次对权重向量的更新较慢。

随机梯度下降是针样本数目较大的时候提出的，根据一个随机样本点去更新权重向量，更新速度上会明显比批剃度下降要快，不过由于只考虑了一个样本，而不去考虑其他样本点，也会导致每次计算的损失函数偏向这个样本，而不是偏向整个样本集。所以在收敛上会比较曲折，最后迭代也不能真正到达最优解，只能逼近。（而且效果是比较随机的，需要我们多次验证之后保存较好的预测向量）

|--------------- 如有优化，重复 1，2 步的展示，分析优化后结果 -------------|