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	**M1**	专业（方向）	**移动信息工程（互联网方向）**
学号	**15352102**	姓名	**韩硕轩**

一、 实验题目

逻辑回归（LR）

二、 实验内容

1. 算法原理

逻辑回归是建立在 PLA 算法基础之上的。在 PLA 中，我们把 w 和每一个文本 x 做点乘，得到的结果和某一个阈值比较，大于阈值就预测为 1，否则预测为-1。在这次实验中，除了负类从-1 变为了 0 以外，预测方式也发生了变化。

由于简单的点乘会使某个类发生的概率超出 0 到 1 的范围，所以我们将这个点乘的值赋给几率比的对数，于是有了以下公式：

$$\log\left(\frac{p}{1-p}\right) = w_0 + \sum_{j=1}^{d} w_j x_j + u$$
$$= \tilde{\mathbf{W}}^{\mathrm{T}}\tilde{\mathbf{X}}$$

化简这个公式，概率 p 就可以用下面这个公式表达：

$$\frac{e^{\tilde{\mathbf{W}}^{\mathrm{T}}\tilde{\mathbf{x}}}}{1+e^{\tilde{\mathbf{W}}^{\mathrm{T}}\tilde{\mathbf{x}}}}$$

由于我们的题目是二分类，所以 p 在 0.5 到 1 之间，文本时正类概率较大，所以预测为 1，反之预测为 0。

这是对于一个文本的预测，如果要对一个数据集预测，我们就要找到一个 w 使得它能预测尽可能多正确的文本。这时我们引入似然函数。整个数据集的似然函数为：

$$\prod_{i=1}^{n} (p_i)^{y_i} (1-p_i)^{1-y_i}$$

我们要使得这个似然函数最大，就相当于给它取对数再取相反数要最小：

$$C(\tilde{\mathbf{W}}) = -L(\tilde{\mathbf{W}}) = -\sum_{i=1}^{n} \left(y_i \log p_i + (1-y_i)\log(1-p_i) \right)$$

这个函数叫做代价函数。它的自变量是 w，要使它最小，也就是要对 w 求导，然后找到导数为 0 或者尽可能为 0 的点。代价函数求完导之后就是梯度。我们在代码中使用迭代的方法使梯度下降越来越小，从而也就能找到越来越小的代价函数，进而找到越来越大的似然函数和对应的 w。

2. 伪代码

1. //Logistic Regression
2. //main part
3. initialize w, eta
4. for I in max_iteration:
5. sum ← [0, 0, …, 0]
6. for j in train_set:
7. p ← $\frac{e^{w^T x_j}}{1+e^{w^T x_j}}$
8. sum ← sum + eta*$(p - y_j)$*X_j
9. l ← $-\log(p) * y_j - \log(1-p) * (1-y_j)$
10. w ← w – s
11. if w == 0:
12. terminate loop i
13. if l < 0.05:
14. eta ← 0.001

3. 关键代码截图（带注释）

初始化 w 和学习率 eta：

```
w = np.random.rand(1, col)[0] #初始化w
eta = 0.01                    #初始化eta
```

在遍历训练集的时候，计算每一个文本的梯度和代价函数：

```
z = np.dot(ts[i], w)           #每个文本和w点乘
if z > 700:
    p = 1                      #防止超出乘方上限
else:                          #不超上限就算logit函数
    p = math.pow(math.e, z) / (1 + math.pow(math.e, z))
t = p - label[i]
```

```
s += eta * t * ts[i]                    #累加梯度，下一行是累加代价函数
l += (-math.log(p, math.e)*label[i]-math.log(1-p, math.e)*(1-label[i]))
```

如果梯度为 0 就跳出循环，否则更新 w，以及视情况更新学习率：

```
if max(s) == 0 && min(s) == 0: #梯度为0，跳出循环
    break;
w -= s  #更新w
if abs(l - l_old) < 0.05:      #代价函数下降缓慢，表示接近最优值了，减小学习率
    eta = 0.001
```

下面就是验证集和测试集的处理了，不是很关键就不截图了。

4. 创新点&优化（如果有）

第一个优化是在 python 中使用了向量化运算（截图是其中一个，其余在上一小题中可见）：

```
z = np.dot(ts[i], w)
```

第二个优化是使用学习率动态下降的方法，在代价函数函数变化小于 0.05 的时候将学习率改成 0.001：

```
if abs(l - l_old) < 0.05:
    eta = 0.001
```

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）

采用小的数据集，各个向量值为图所示：

	A	B	C
1	2	3	1
2	1	4	1
3	5	2	0

W 初始为全 1，学习率为 0.1。手算得对于第一个向量做完之后，XW^T为 6，进而第一维应减去 0.000247。第二个第三个向量同理。所以 W 应减去向量（-0.000247，-0.000247, 0.099）。之后两个向量也是一样的方法计算，得到W一共应减去（0.099， 0.499， 0.198）。打印出来的结果（前五次）可以验证正确性：

```
After 1 iteration(s), w is [ 0.90052806  0.50090946  0.80179791] and -log(likelihood) is [ 8.00528678]
After 2 iteration(s), w is [ 0.80350683  0.00787091  0.61105737] and -log(likelihood) is [ 5.03861899]
After 3 iteration(s), w is [ 0.7250582  -0.41897458  0.4681693 ] and -log(likelihood) is [ 2.29031696]
After 4 iteration(s), w is [ 0.71745387 -0.56233961  0.49485338] and -log(likelihood) is [ 0.84974725]
After 5 iteration(s), w is [ 0.72854447 -0.62540422  0.56359283] and -log(likelihood) is [ 0.69105819]
[Finished in 0.1s]
```

看到第一次迭代 W 确实减去了我刚刚算出来的那个向量。且在前五次迭代中，代价函数一直在下降，且下降得越来越慢，说明梯度越来越小。

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 8000 个训练集数据的最后 1000 个用来做验证集。

在学习率为 1 的时候分别迭代 10/100/1000 次的结果如下（没有动态改变学习率）：

```
0.555
[Finished in 2.6s]
```
```
0.659
[Finished in 6.3s]
```
```
0.662
[Finished in 42.0s]
```

在学习率为 0.1 的时候分别迭代 10/100/1000 次的结果如下（没有动态改变学习率）：

```
0.558
[Finished in 2.7s]
```
```
0.659
[Finished in 6.2s]
```
```
0.661
[Finished in 40.3s]
```

在学习率为 0.001 的时候分别迭代 10/100/1000 次的结果如下（没有动态改变学习率）：

```
0.674
[Finished in 2.9s]
```
```
0.735
[Finished in 6.5s]
```
```
0.737
[Finished in 40.3s]
```

后来采用动态改变学习率的方法，使用 0.1 的学习率迭代 10/100/1000 次结果如下：

```
0.583
[Finished in 2.7s]
```
```
0.757
[Finished in 6.7s]
```
```
0.633
[Finished in 45.3s]
```

可见：①在一定范围内，学习率越小，结果正确率越高；②大部分情况下迭代次数越多，结果正确率越高（小部分情况包括一开始的震荡和动态改变学习率之后）；③动态改变学习率对结果有优化作用。

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？

答：可能迭代会停不下来，因为梯度为 0 可能永远也达不到。

2. 学习率的大小会怎么影响梯度下降的结果？

答：学习率太大可能会导致收敛速度变慢，甚至发散。对应到抛物线的图中，目标点

可能从左半轴和右半轴之间来回跳，甚至跳出去。

我用第三大题中自己写的小数据集做了测试，在学习率较大（我设置的是 1）的情况下，一开始的梯度出现明显的震荡：

```
After 1 iteration(s), w is [-0.49207911 -6.48635807 -1.97303141] and -log(likelihood) is [ 8.00528678]
After 2 iteration(s), w is [ 2.50792037 -1.9863586   8.52696649] and -log(likelihood) is [ 34.25445262]
After 3 iteration(s), w is [ 1.00801894 -9.48586571  5.52716364] and -log(likelihood) is [ 9.63012604]
After 4 iteration(s), w is [ 2.20704208 -7.08782125  9.12423485] and -log(likelihood) is [ 1.60618219]
After 5 iteration(s), w is [ 2.20704192 -7.08782297  9.12423484] and -log(likelihood) is [  5.15174775e-07]
[Finished in 0.1s]
```

而将学习率设为 0.1 时就迅速收敛：

```
After 1 iteration(s), w is [ 0.90052806  0.50090946  0.80179791] and -log(likelihood) is [ 8.00528678]
After 2 iteration(s), w is [ 0.80350683  0.00787091  0.61105737] and -log(likelihood) is [ 5.03861899]
After 3 iteration(s), w is [ 0.7250582  -0.41897458  0.4681693 ] and -log(likelihood) is [ 2.29031696]
After 4 iteration(s), w is [ 0.71745387 -0.56233961  0.49485338] and -log(likelihood) is [ 0.84974725]
After 5 iteration(s), w is [ 0.72854447 -0.62540422  0.56359283] and -log(likelihood) is [ 0.69105819]
[Finished in 0.1s]
```

迭代 100 次之后发现代价函数一直在减小，到一个比较稳定的水平，此时的梯度也趋于 0。

```
After 90 iteration(s), w is [ 0.98316383 -1.70366508  1.911799  ] and -log(likelihood) is [ 0.0615844]
After 91 iteration(s), w is [ 0.98439502 -1.70845759  1.91797624] and -log(likelihood) is [ 0.0609481]
After 92 iteration(s), w is [ 0.98561382 -1.71320071  1.92409019] and -log(likelihood) is [ 0.0603249]
After 93 iteration(s), w is [ 0.98682048 -1.71789544  1.93014216] and -log(likelihood) is [ 0.05971439]
After 94 iteration(s), w is [ 0.98801523 -1.72254278  1.9361334 ] and -log(likelihood) is [ 0.05911619]
After 95 iteration(s), w is [ 0.98919831 -1.72714367  1.94206512] and -log(likelihood) is [ 0.05852993]
After 96 iteration(s), w is [ 0.99036994 -1.73169905  1.9479385 ] and -log(likelihood) is [ 0.05795524]
After 97 iteration(s), w is [ 0.99153035 -1.7362098   1.95375469] and -log(likelihood) is [ 0.0573918]
After 98 iteration(s), w is [ 0.99267976 -1.74067681  1.95951479] and -log(likelihood) is [ 0.05683927]
After 99 iteration(s), w is [ 0.99381836 -1.74510092  1.96521988] and -log(likelihood) is [ 0.05629734]
After 100 iteration(s), w is [ 0.99494636 -1.74948295  1.97087101] and -log(likelihood) is [ 0.0557657]
[Finished in 0.2s]
```

3. 批梯度下降和随机梯度下降的优缺点？

答：批梯度下降每次迭代都会考虑所有的样本，这样的一次迭代下来得到的是一个标准的梯度值，更新幅度较大。但是由于每次都要遍历所有的样本，所以算法时间复杂度高一些。而随机梯度下降每次只针对一个样本，随机性比较大，而且容易陷入局部最优，即得到的最终的 w 可能正确率并不高。但是这样更新速度会比较快。