**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**

# 实验题目

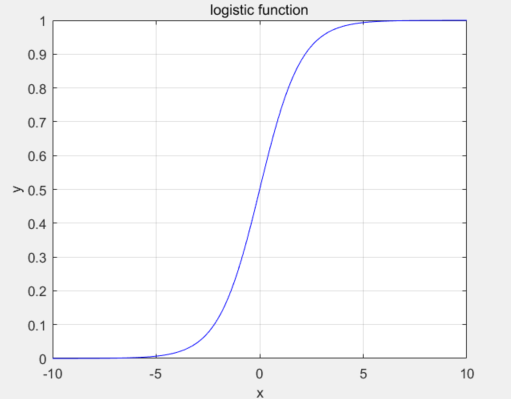
1. **实现Logistic回归**

# 实验内容

1. 算法原理

Logistic回归主要用于二分类，是一种非线性回归模型。目的就是从训练集训练出一个0/1分类模型。不过Logistic回归是一种软分类，需要算出属于每个分类的概率。

Logistic回归的主要判决做法和PLA差不多，都是使用一个进行判决分类，其中为自变量，即数据的特征值，为预测结果。与PLA不同的是使用了Logistic(sigmoid)函数作为预测函数，因为其将加权分数映射到另一个更合理的数据空间，使加权分数的大小能够反映概率的大小。



函数在x=0的时候取值为1/2，而随着x逐渐变小，函数值趋于0，说明该数据属于正类别的概率为 0；x逐渐变大，函数值逐渐趋于1，说明该数据属于正类别的概率为 1。这是一个正常的概率区间。

之前提到了其与PLA的差别主要在于预测函数改变了，对比如下：

|  |  |
| --- | --- |
| PLA | Logistic回归 |
|  |  |

那么意味着我们训练最终的到函数为：

其中判决阈值，此公式化简一下：

此时。

为hypothesis函数，算的是属于正类的概率，属于负类别的概率为，即

此时与PLA类似我们还是要训练出。此时需要用到最大似然估计法。首先对于单个样本，其后验概率为：

此时最大似然函数为：

那么此时要找到一组模型参数，使得最大似然函数最大，我们对其取对数，此时称其为对数似然函数：

此时最大似然估计目的就是要使对数似然函数取到最大值，其实这里可以使用梯度上升法求解。不过我们还是取负，然后使用梯度下降法求解，那么此时定义损失函数J：

此时使J函数最小，也能到达最大似然的目的。那么此处就需要使用梯度下降法进行求解。

梯度下降法可以理解为：我们在山上不知道怎么下山，于是决定走一步算一步，在每走到一个位置的时候，求解当前的梯度，沿着梯度的最大值，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度，再向当前最陡峭的位置向下走一步。这样一步步的走下去会到达到山脚。当然也可能我们不能走到山脚，而是到了某一个局部的山峰低处。

那么此时我们更新w就需要使得w的每一维度都沿着梯度最大地方进行更新，直至收敛，那么此时我们需要先求解最大的梯度方向，那么我们需要对之前定义的损失函数J求导，j表示第j维，i表示第i个样本。

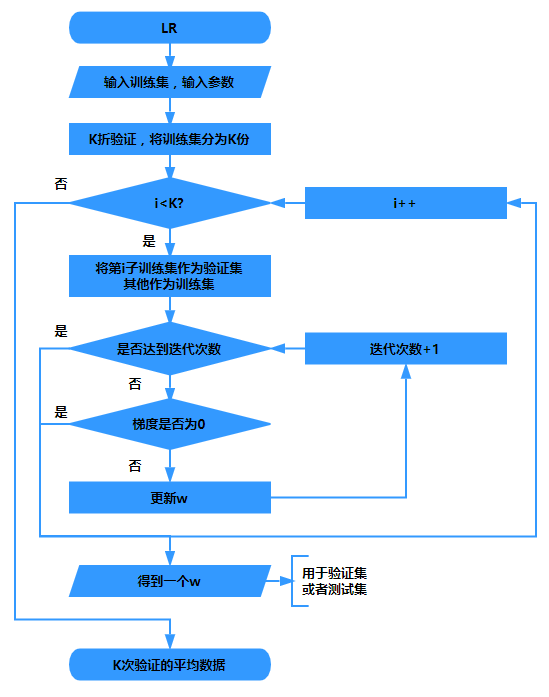
我们只需要求解：

将其带入（1）中得：

此时我们得到的更新公式，其中为步长：

然后我们只要更新w的所有维度即可完成一次w的更新。最终迭代至逼近最优解或者收敛时停止。

1. 伪代码&流程图
   1. 总流程



* 1. 训练部分伪代码

|  |
| --- |
| 设置初始w  **for** 迭代次数 = 1 to 设置迭代次数  **if** 梯度 = 0 **then return** w  **else** 使用公式更新w  **end**  **return** w |

1. 关键代码截图（带注释）
   1. 迭代更新w部分，eps为matlab中自带一个近似0的数eps=2.22044e-16。

|  |
| --- |
| **for** i = 1 : iter %迭代  %计算损失函数  Err = ((1 ./ (1 + exp(-w \* tmp\_train')) - sign\_train') \* tmp\_train) / train\_row;  **if** Err < eps  **break**;  **end**  step = step \* delta;%步长动态减少  w = w - step \* Err;%更新w  **end** |

* 1. 判决部分

|  |
| --- |
| **function** test\_ans=logistic\_test(w,test)%判决函数  test\_ans = 1 ./ (1 + exp(-w \* test'))';  test\_ans(test\_ans >= 0.5) = 1; %0.5为判决线  test\_ans(test\_ans < 0.5) = 0;  **end** |

1. 创新点&优化
2. **向量化运算**

此处选择的还是matlab，因为矩阵计算速度比c++快得多，跟PLA一样减少了代码实现中减少很多for循环包括在对测试指标计算和预测的时候都做到不用for。只有K折和迭代用了。速度纯计算的话大概3s能迭代1w次。

此处说明一下指标的统计方式，如下表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指标 | 说明 | 实际\*10+预测 | | S |
| TP | 本来为+1，预测为+1 | 1\*10 | 1 | 11 |
| FN | 本来为+1，预测为0 | 1\*10 | 0 | 10 |
| TN | 本来为0，预测为0 | 0\*10 | 0 | 0 |
| FP | 本来为0，预测为+1 | 0\*10 | 1 | 1 |
| S=sign(tmp\_val \* w') + 10 \* sign\_val; %原理见上表  TP = sum(S( : ) == 11);  FN = sum(S( : ) == 9);  TN = sum(S( : ) == -11);  FP = sum(S( : ) == -9);  Accuracy = (TP + TN) / (TP + FP + TN + FN);  Recall = TP / (TP + FN);  Precision = TP / (TP + FP);  F1 = (2 \* Precision \* Recall) / (Precision + Recall); | | | | |

1. **随机梯度下降**

随机梯度下降是通过每个样本来迭代更新一次，可以最快的将w迭代到最优解。每次迭代只是考虑让该样本点的代价函数趋向最小，而不管其他的样本点，这样算法会很快，但是收敛的过程会比较曲折，其得到的并不是准确的一个梯度，容易陷入到局部最优解中。

公式为：

相应的代码中公式部分改为,pos为任意一行的下标

|  |
| --- |
| Err = ((1 / (1 + exp(-w \* tmp\_train(pos, : )')) –  sign\_train(pos)') \* tmp\_train(pos, : )) / train\_row; |

1. **正则化**

正则化是结构风险最小化策略的实现，是在经验风险上加一个正则化项或惩罚项，消除高次项的影响。正则化项一般是模型复杂度的单调递增函数，模型越复杂，正则化项就越大。

此时代价函数J为,引入一个系数1/2，方便之后求导。

所以此时更新公式变为：

是正则项系数，如果它的值很大，说明对模型的复杂度惩罚大，对拟合数据的损失惩罚小，这样它就不会过分拟合数据，在训练数

据上的偏差较大，在未知数据上的方差较小，但是可能出现欠拟合的现象。如果它的值很小，说明比较注重对训练数据的拟合，在训练数据上的偏差会小，但是可能会导致过拟合。

所以此时代码变为：

|  |
| --- |
| Err = ((1 ./ (1 + exp(-w \* tmp\_train')) - sign\_train')  \* tmp\_train + lambda \* w) / train\_row; |

动态步长，标准化就不解释了，动态步长每次乘一个使得步长减少，标准化matlab中自带zscore函数直接用就ok

# 实验结果及分析

1. 实验结果展示示例

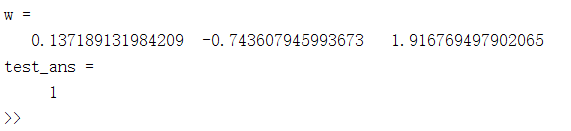
小数据集测试：

|  |  |  |  |
| --- | --- | --- | --- |
|  | X1 | X2 | Label |
| A | 1 | 2 | 1 |
| B | 2 | -1 | 0 |
| C | 3 | 3 | ? |

* 初始，步长为1.
* 增广特征向量：
* 增广权向量的转置乘以增广特征向量:
* 计算每一维度梯度：(此处就省去1/m的平均，方便计算)
* 更新w
* 此时

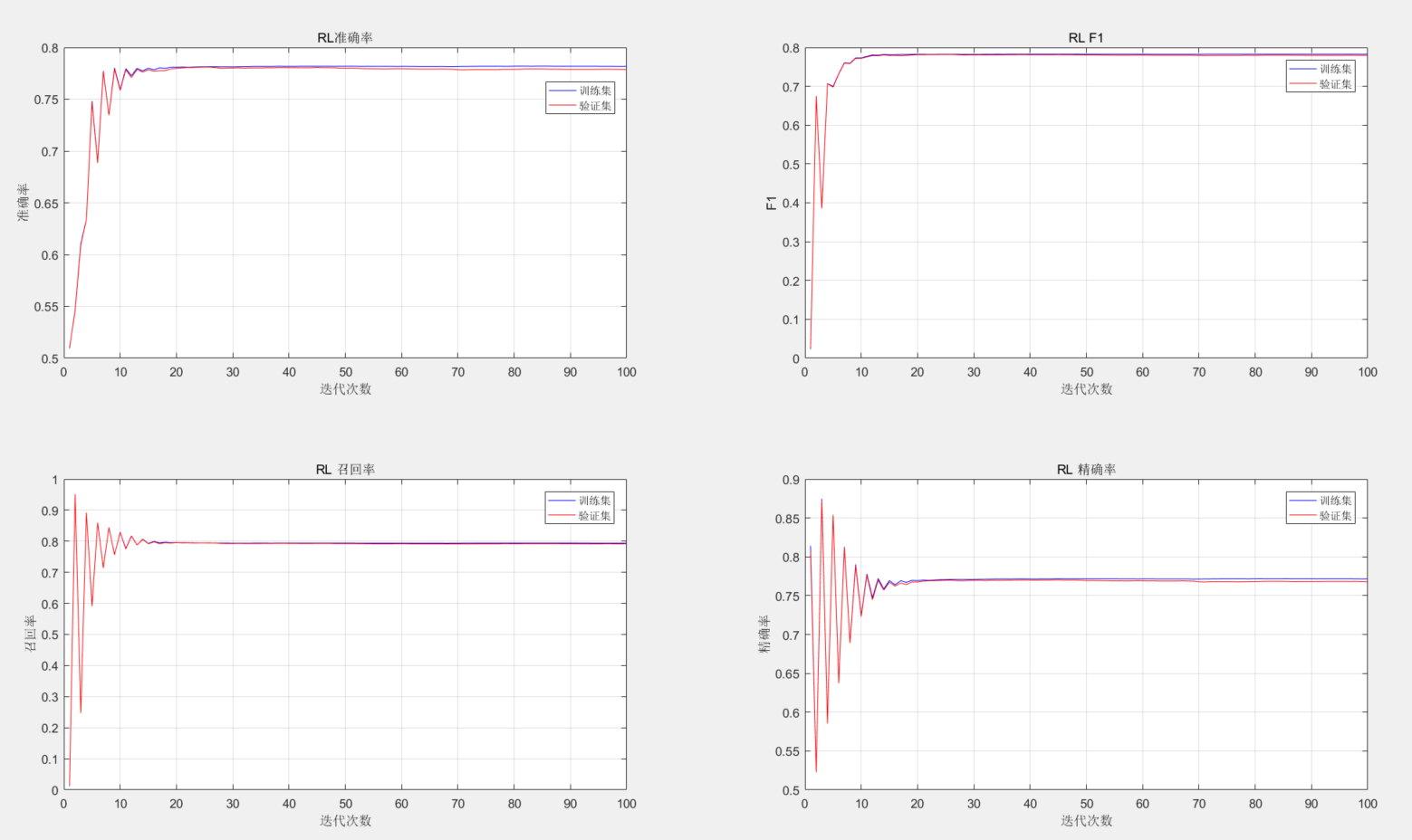
所以此时C判定为1。

* 此时我们按照w全1迭代1次进行测试。

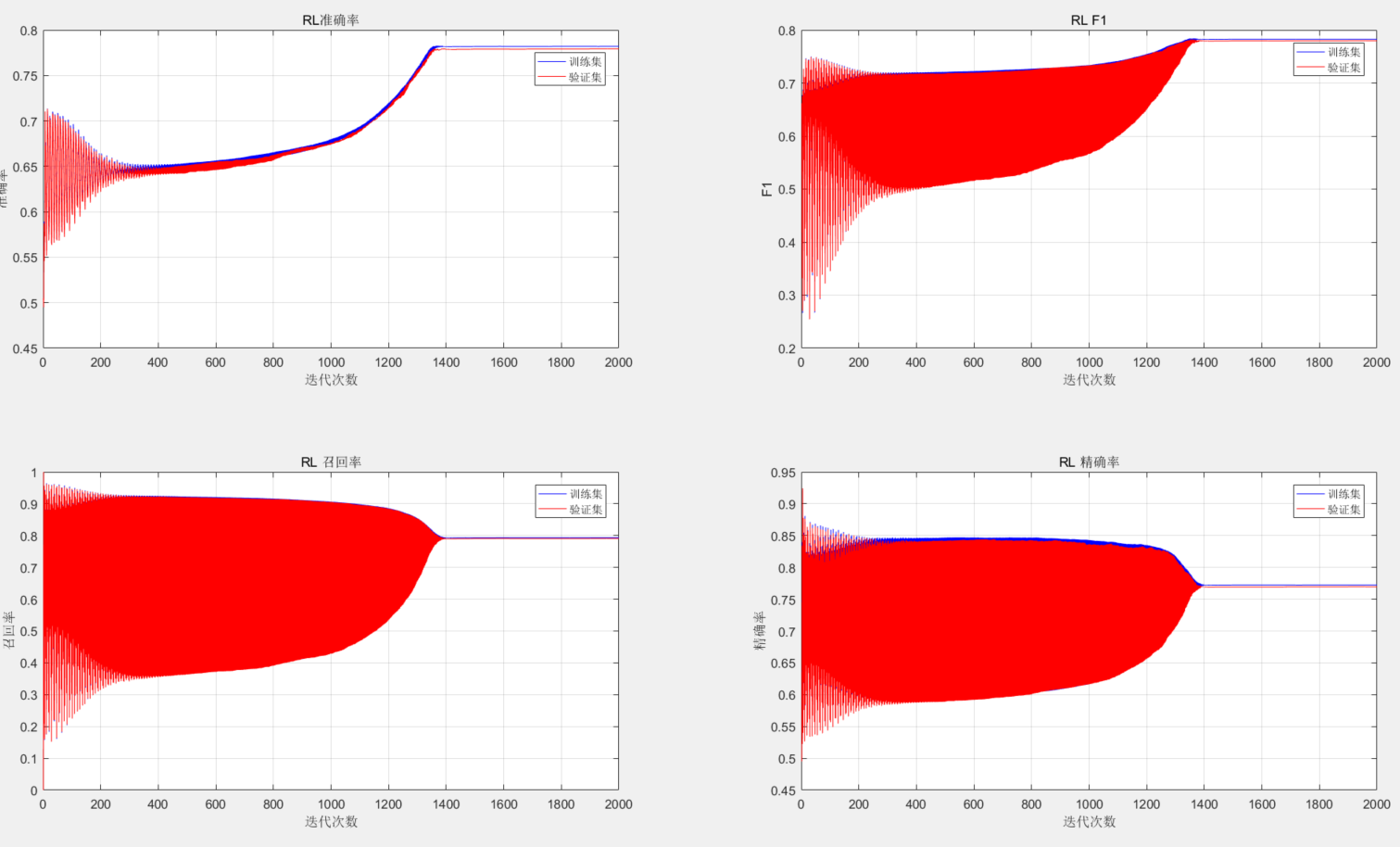


结果和理论计算一致。

1. **评测指标展示即分析**
   1. **无任何优化，使用10折交叉验证**

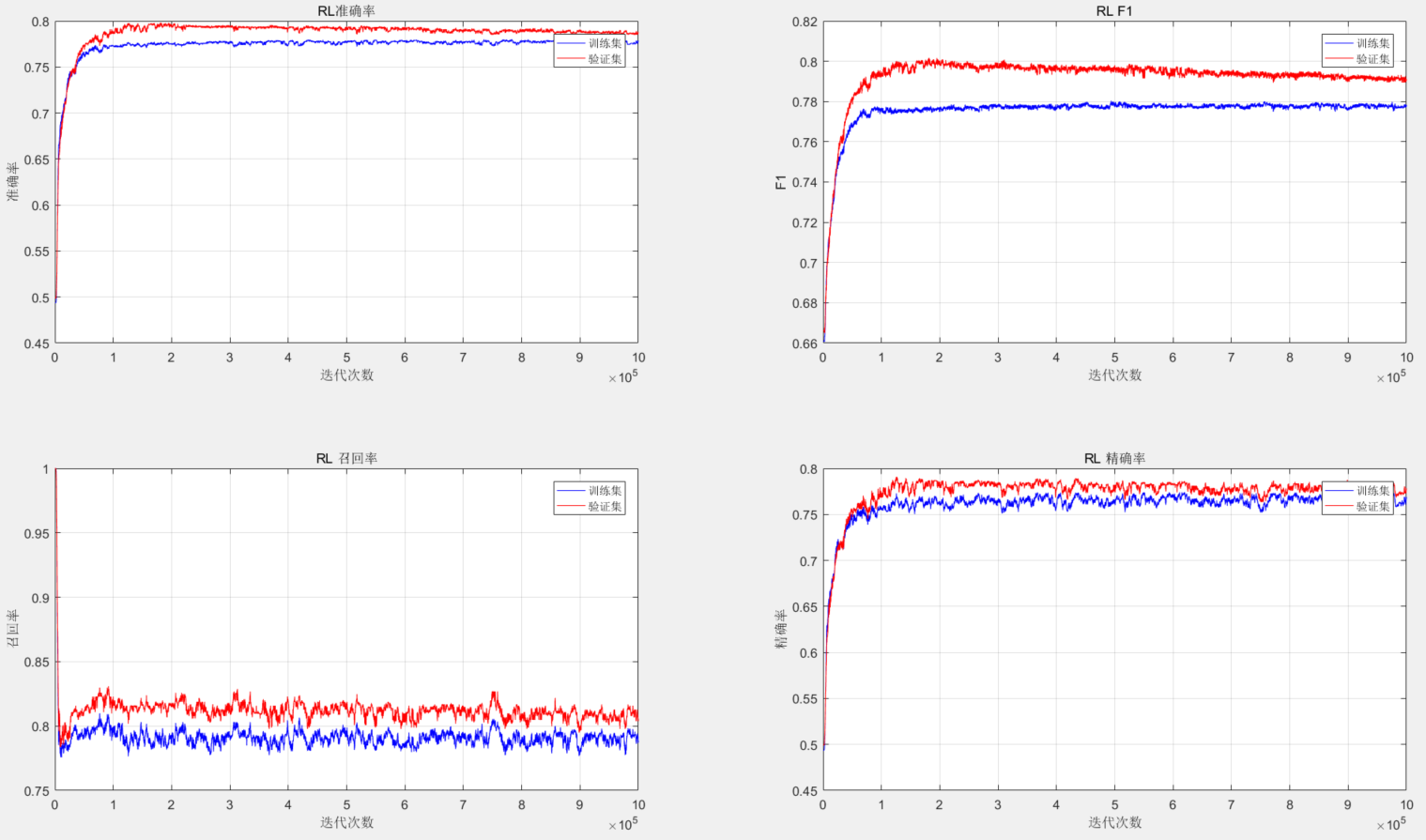


* 1. **动态步长，同时修改迭代次数**



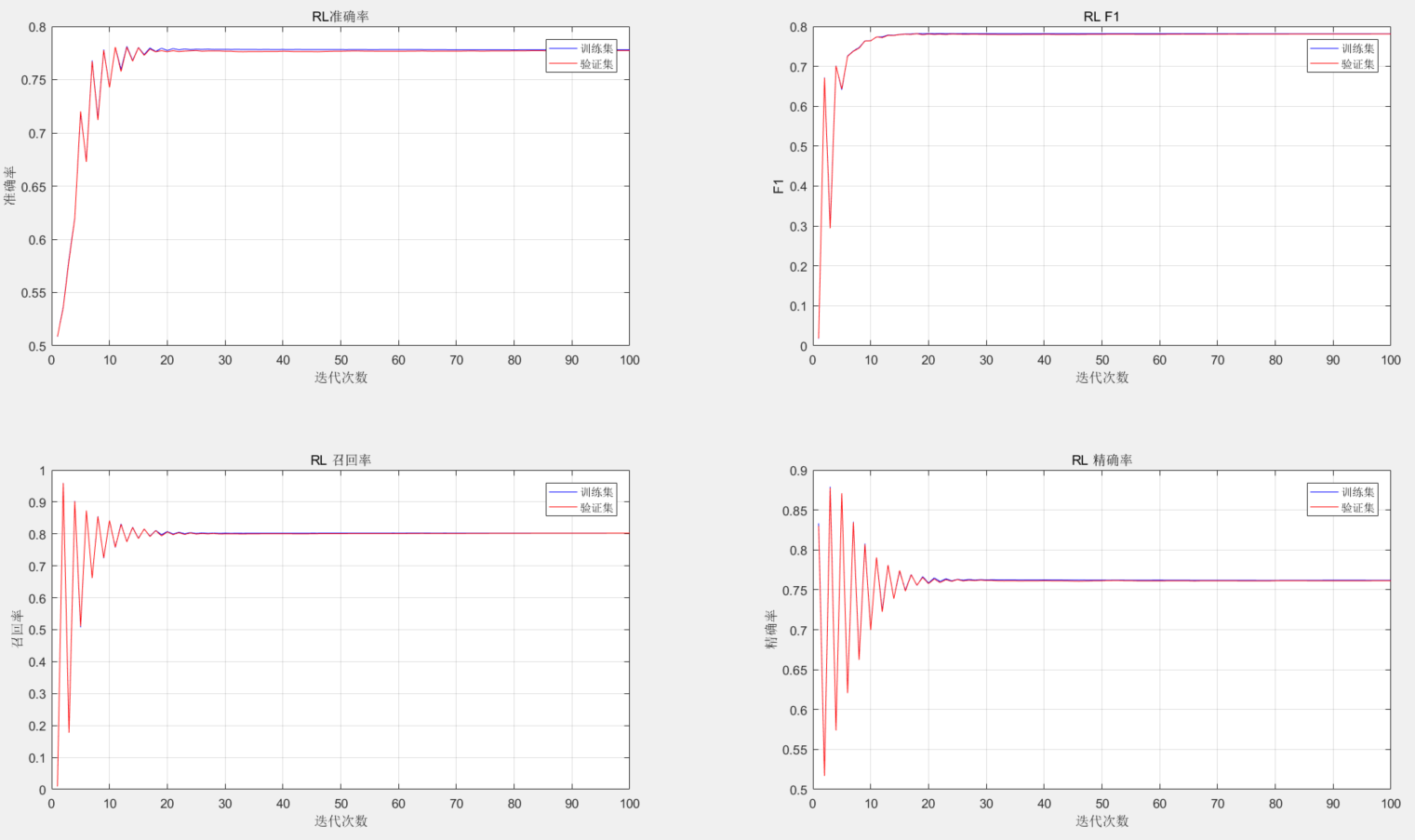
除了肉眼不可见的小幅提升之外没有多少影响，主要就是因为之前无优化的时候步长只有0.1，本身不会出现无法收敛的情况，就是此处最后收敛的时候步长更小，最终只有其1/100。可以看到其收敛的时候准确率是缓慢上升的，不是之前那种快速上升状态

* 1. **随机梯度下降法**



随机梯度下降法迭代速度是批量梯度下降法的10倍不止，此时的时间基本都花在计算画图上了，但是实际我们可以看到随机梯度下降法在最优是比批量梯度下降法的。

* 1. **正则化，10折交叉验证。**



可以看到这个训练集和验证集的指标基本上是全程重叠，说明其对于过拟合线性还是有消除作用的，但是这个数据集本身在我们测试的时候就发现不会产生什么严重的过拟合。所以正则化效果只有一点点。

# 思考题

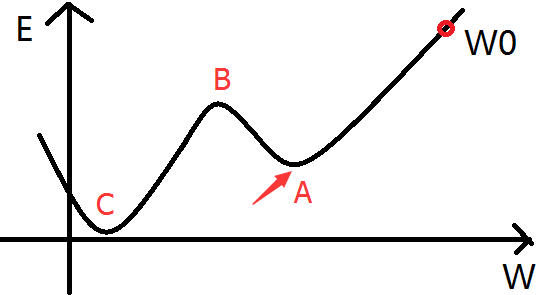
* + - 1. **如果把梯度为 0 作为算法停止的条件，可能存在怎样的弊端？**

当数据集大的时候，基本不存在能够有限的迭代次数内使得梯度为0。如果步长固定，基本上最终结果只能是接近0，如果此时用0作为停止条件，有可能最终停不下来只会在0附近震荡，造成算法的死循环。动态改变步长最终也只能是慢慢接近0，但是要进行更多次的迭代但是此时会消耗极大的时间。

算法中我们可以考虑使用一个近似0的数作为停止迭代的条件，再加上动态步长作为停止的条件而不是以完全为0作为停止条件。

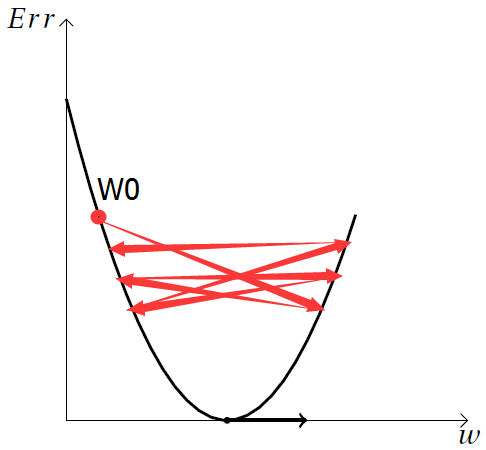
* + - 1. **𝜂 的大小会怎么影响梯度下降的结果？**

过小：导致收敛过慢，想要达到收敛需要大量迭代，所需的时间将会很大。同时其很容易陷入局部最优。如下图：



容易因为不足直接在箭头所指A处停止收敛，无法跨过B达到全局最优C。

过大：导致无法收敛到最优，会出现震荡。如下图：



此时会无法到达最优值会在处震荡，根据步长的大小震荡的幅度也会随之变化。例如一开始写的时候步长为1(没有取平均)，准确率震荡幅度就在0.5~0.8处震荡。

* + - 1. **批梯度和随机梯度的优缺点？**

|  |  |  |
| --- | --- | --- |
|  | 优点 | 缺点 |
| 批量梯度下降法 | 使用所有训练样本计算w，最终能得到全局最优解。 | 计算每一个w值都需要遍历计算所有样本，当数据量大的时候这是比较费时的计算。 |
| 随机梯度下降法 | 每次只用一个训练样本计算w，计算量少，速度快。 | 迭代之后是向着全局最优解前进的，但是结果往往在最优解附近，同时也很容易收敛到局部最优。 |