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	**1511**	专业（方向）	**软件工程（移动信息）**
学号	**15352237**	姓名	**刘顺宇**

一、 实验题目

Logistic Regression Model

二、 实验内容

1. 算法原理

1.1. 简述：逻辑回归是一种软分类模型，即概率模型，对每个分类求概率后取概率最大的分类，主要思路是通过计算数据权重，根据权重了解预测目标的可能性。逻辑回归的实现思路建立在 PLA 模型之上的，在 PLA 模型中只能实现解决线性可分问题，而逻辑回归模型通过 logistic（sigmoid）函数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，从而能够完成对事件发生的概率进行预测。它比直接的硬分类模型多了一个概率预测，因此更加有现实意义。

1.2. 理论推导：

1.2.1. 对于一个软分类问题，目标函数定义如下：

$$f(x) = P(label|x) \in [0,1]$$

即在给定特征向量 x 的情况下，属于 label 类的可能性多大

1.2.2. 特征向量的每一个维度，都会对结果产生影响，那么与 PLA 一样，可以模拟一个带权重的分数：

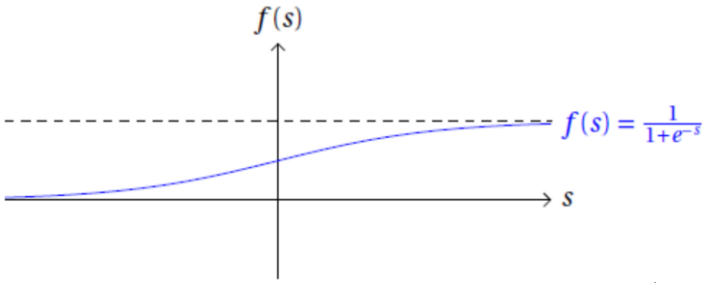
$$s = \sum_{t=0}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

注意这里 t 从 0 开始，这是由于预测时需要一个阈值进行判断，而我们将阈值作为w_0，然后对$\boldsymbol{x}$向量进行向量扩展，使得$x_0 = 1$即可。表达式中的 w_i 表示第 i 维特征的权重，$w_i > 0$ 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

1.2.3. 既然对于软分类来说，要算出属于每个分类的概率，而我们之前所学习过的模型均属于硬分类模型，即结果非此即彼，无法知道相关概率，所以需要一个新的决策函数，利用某种函数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，在逻辑回归里使用的是：logistic（sigmoid）函数

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

将数据从（$-\infty$,∞）映射到（0,1）



logistic（sigmoid）函数的特征：

$\theta(-\infty) = 0$　当加权分数无穷小，该数据属于正类别的概率为 0

$\theta(0) = 0.5$　当加权分数为 0，该数据属于正/负类别的概率为 0.5，即该数据属于任一类别的概率相同

$\theta(+\infty) = 1$　当加权分数无穷大，该数据属于正类别的概率为 1

1.2.4. 利用 logistic 函数，我们可以构成一个新的假说模型：

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

要求解的是 w，根据上面的假说模型，h(x) 算得的是属于正类的概率，属于负类别的概率即为 1 - h(x)，当 h(x) 大于 0.5 的时候，说明该数据更大可能属于正类别。

1.2.5. 那么我们可以把最开始提及的目标函数 f(x) 与 h(x) 联合起来:

$$f(x) = P(label|x) = h(x)^y \big(1-h(x)\big)^{1-y}$$

y 表示 x 对应的分类标签

当 $y = 1$，$f(x) = P(label|x) = h(x)$

当 $y = 0$，$f(x) = P(label|x) = 1 - h(x)$

1.2.6. 用贝叶斯派的观点来看待这个问题，不同的参数设置代表着不同的模型，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），考虑整个数据集，似然函数如下:

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

根据最大似然估计算法，要找到一组模型参数，使得上式最大

对 likelihood 取对数，再取负数之后，即可得到以下的函数：

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

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

对以上的函数取最小，即达到最大似然的目的

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

1.2.7. 对 w 求导，利用梯度下降法，通过不断地迭代使 w 逼近最优解直至收敛

$$\text{Repeat: } \tilde{\mathbf{W}}_{new}^{(j)} = \tilde{\mathbf{W}}^{(j)} - \eta \frac{\partial C(\tilde{\mathbf{W}})}{\partial \tilde{\mathbf{W}}^{(j)}}$$
$$= \tilde{\mathbf{W}}^{(j)} - \eta \sum_{i=1}^{n} \left[\left(\frac{e^{\tilde{\mathbf{W}}^{\mathrm{T}} \tilde{\mathbf{X}}_i}}{1+e^{\tilde{\mathbf{W}}^{\mathrm{T}} \tilde{\mathbf{X}}_i}} - y_i \right) \tilde{\mathbf{X}}_i^{(j)} \right]$$
$$\text{Until convergence}$$

η 表示学习率，j 表示第几维，i 表示第几个样本

1.3. 算法流程：

输入：特征向量集合 {x}，标签集合 {y}

输出：最优解 w

1.3.1. 初始化 w_0

1.3.2. 利用梯度下降法更新 w

1.3.3. 直至梯度为 0 或者迭代足够多次

1.3.4. 利用最优 w 来预测测试集特征向量所对应的标签，计算属于正/负类别的概率

1.4. 优化思路：

1.4.1. 学习率 η：

动态学习率：初始学习率较大，当梯度下降到接近最优值时，将学习率降低

通过验证集的方式确定学习率：设置不同的学习率，如果模型都可以较好拟合数据，选择该模型为最优模型，用于测试集的预测。

1.4.2. 随机梯度下降：原本的更新权重向量公式使用批梯度下降，每次更新权重向量都考虑所有样本，而随机梯度下降每次更新参数都只考虑一个样本

1.4.3. 标准化：对数据进行预处理操作。

1.4.4. 正则化：我们并不会事先知道要减小哪一个参数的值。但是，一般来讲参数的值越小，通常对应更加简单的函数，就不容易发生过拟合的问题。因此，我们通常在损失函数中惩罚比较大的参数，以得到更为简单的模型。通过增加正则化项，惩罚较大的参数。

$$-\sum_{i=1}^{M} y_i h(x_i) + (1-y_i)\big(1-h(x_i)\big) + \lambda \sum_{j=1} W_j^2$$

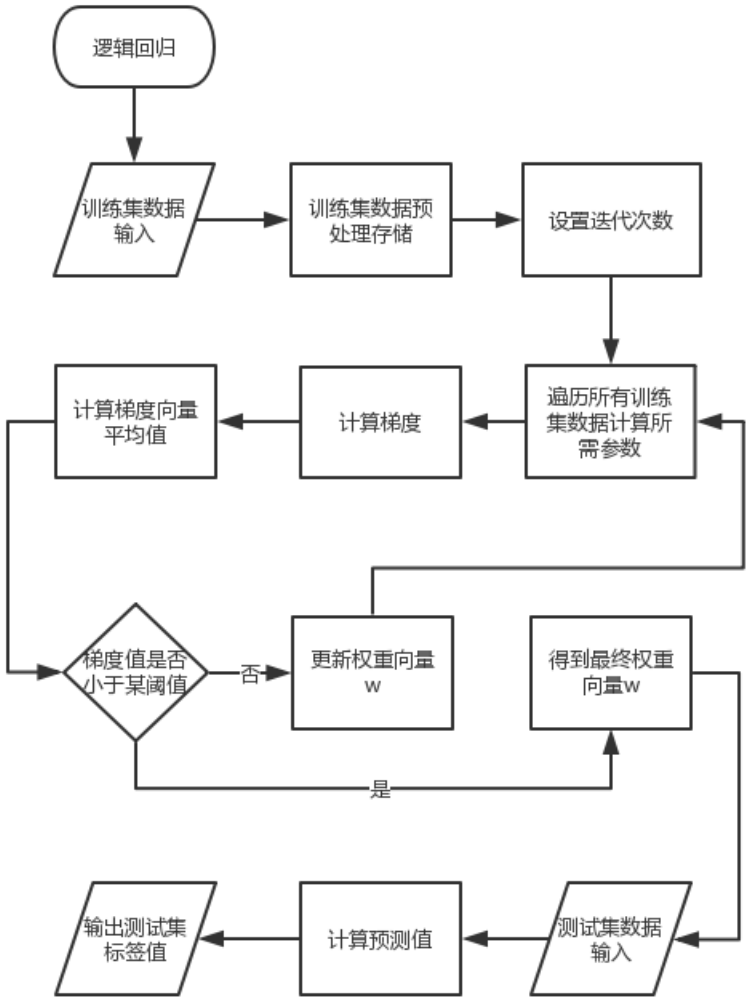
理论推导：对上式求导代入权重向量更新公式得

$$W_{new}^{(j)} = W^{(j)} - \mu \left\{ \sum_{i=1}^{n} \left[\left(\frac{e^{W^T X_i}}{1+e^{W^T X_i}} - y_i \right) X_i^{(j)} \right] + 2\lambda W^{(j)} \right\}$$

2. 伪代码流程图

2.1. 普通逻辑回归算法代码流程图（具体代码思想在关键代码解析中讲诉）

首先读取训练集进行处理，然后选用各种不同的逻辑回归模型进行迭代利用梯度下降法计算权重向量，最后通过权重向量来对测试集数据进行预测。



3. 关键代码截图（这里只解释最关键的建树过程，其他具体代码可看 cpp 文件，有详尽注释）

3.1. 普通逻辑回归

首先我们需要设置迭代次数，这个迭代次数是我们迭代更新权重的上限，如果我们没办法在后面根据预先设好的判断退出迭代，则根据迭代次数上限来退出迭代，避免陷入死循环。

接着我们根据梯度公式计算梯度，然后计算当前梯度向量的平均值，然后根据我们预先设置好的阈值来判断当前迭代是否已经到达最优值，如果低于阈值则退出迭代，得到最终的权重向量 w。

```
118     //普通逻辑回归
119     if (method == 1) {
120         int iter = 5000;//设置迭代次数
121         double alpha = 0.0001;
122         while (iter--) {
123             //动态步长
124             alpha = double(iter) * 0.0000002;
125 
126             //计算梯度
127             double wx[10000];
128             for (int i = 0; i < Train_row; i++) {
129                 wx[i] = vector_mul(w, Train_set[i].data, Train_col);
130             }
131             double w_new[50];
132             double gradient_avg = 0;
133             for (int j = 0; j < Train_col; j++) {
134                 double gradient = 0;
135                 for (int i = 0; i < Train_row; i++) {
136                     double logistic = exp(wx[i]) / (1 + exp(wx[i])) - Train_set[i].value;
137                     gradient = gradient + logistic * Train_set[i].data[j];
138                 }
139                 w_new[j] = w[j] - alpha * gradient;
140                 gradient_avg += abs(gradient);
141             }
142 
143             //计算梯度向量平均值
144             gradient_avg = gradient_avg / Train_col;
145             if (min_gradient_avg > gradient_avg) min_gradient_avg = gradient_avg;
146 
147             //如果梯度向量平均值小于某阈值则不再迭代
148             if (gradient_avg > 800) {
149                 for (int i = 0; i < Train_col; i++) {
150                     w[i] = w_new[i];
151                 }
152             }
153             else {
154                 break;
155             }
156         }
157         cout << min_gradient_avg << endl;
158     }
```

4. 创新点&优化（为了避免重复，创新点算法没有在关键代码截图中提及）

4.1. 交叉验证法

由于这次实验只给了训练集没有验证集，所以自己去查看了交叉验证法的相应资料并加以实现，基本思路便是先将数据集随机打乱，然后将数据集分为 K 份（在这次实验中分了 8 份），然后交叉验证每次取 K-1 份数据作训练集，1 份作验证集，循环遍历 K 次，得到的 K 个准确率再取平均便得到了最终的准确率。由于代码过于简单只贴训练集读取的代码。

```
/*
训练集数据输入处理
输入：交叉验证法分割分数、训练数据文件
*/
void Train_file_input(int file_split) {
    //初始化各项数据
    Train_row = 0;//记录总行数
    Train_col = 0;//记录总列数

    //读取训练集数据文件
    //ifstream train_file("1.csv");
    ifstream train_file("train.csv");
    int tem_row_split = 0;
    if (train_file) {
        string train_str;
        while (getline(train_file, train_str)) {
            //根据交叉验证的分割份数读取文件
            if (!((file_split * 1000) <= tem_row_split && tem_row_split < ((file_split + 1) * 1000))) {
            //if (true) {
                //记录每行数据的标签值
                int pos = train_str.rfind(',');//反向寻找第一个Tab出现的位置
                string value_str = train_str.substr(pos + 1, train_str.length() - pos - 1);//截取值字符串
                if (value_str == "1") Train_set[Train_row].value = 1;
                else if (value_str == "0") Train_set[Train_row].value = 0;
                else cout << "Error" << endl;

                //记录每行数据的特征值与标签值
                train_str = train_str.substr(0, pos);//截取有效字符串
                char* train_c_str = (char*)train_str.c_str();
                const char* d = ",";//以空格作截取词
                char* tem_c = strtok(train_c_str, d);//截取有效单词
                int tem_col = 1;
                while (tem_c) {
                    //存储数据特征值,判断是否进行减少分叉操作
                    Train_set[Train_row].data[tem_col]= atof(tem_c);
                    tem_c = strtok(NULL, d);
                    tem_col++;
                }
                Train_col = tem_col;
                Train_row++;
            }
            tem_row_split++;
        }
        train_file.close();
    }
```

4.2. 动态步长

初始学习率较大，当梯度下降到接近最优值时，将学习率降低。在这里我们通过设计一条公式使得步长随着迭代次数的增加而增加便实现了动态步长。

```
        while (iter--) {
            //动态步长
            alpha = double(iter) * 0.0000002;
```

4.3. 随机梯度下降法

原本的更新权重向量公式使用批梯度下降，每次更新权重向量都考虑所有样本，而随机梯度下降每次更新参数都只考虑一个样本。而我们选取样本的办法采用了随机选取，使用了随机数随机从训练集数据中选取一条数据计算梯度并更新权重向量。

```
159     //随机梯度下降
160     else if (method == 2) {
161         int iter = 350000;//设置迭代次数
162         double alpha = 0.0001;
163         int tem_row = 0;
164         while (iter--) {
165             //动态步长
166             alpha = double(iter) * 0.000000002;
167 
168             //计算随机梯度
169             double wx;
170             tem_row = rand() % Train_row;
171             wx = vector_mul(w, Train_set[tem_row].data, Train_col);
172             double w_new[50];
173             for (int j = 0; j < Train_col; j++) {
174                 double gradient = 0;
175                 double logistic = exp(wx) / (1 + exp(wx)) - Train_set[tem_row].value;
176                 gradient = gradient + logistic * Train_set[tem_row].data[j];
177                 w_new[j] = w[j] - alpha * gradient;
178             }
179 
180             //更新权重
181             for (int i = 0; i < Train_col; i++) {
182                 w[i] = w_new[i];
183             }
184         }
185     }
```

4.4. 正则化

我们在损失函数中惩罚比较大的参数，以得到更为简单的模型。通过增加正则化项，惩罚较大的参数。

$$-\sum_{i=1}^{M} y_i h(x_i) + (1-y_i)\big(1-h(x_i)\big) + \lambda \sum_{j=1} W_j^2$$

理论推导：对上式求导代入权重向量更新公式得

$$W_{new}^{(j)} = W^{(j)} - \mu\left\{\sum_{i=1}^{n}\left[\left(\frac{e^{W^T X_i}}{1+e^{W^T X_i}} - y_i\right) X_i^{(j)}\right] + 2\lambda W^{(j)}\right\}$$

```
195             //计算正则化梯度
196             double wx[10000];
197             for (int i = 0; i < Train_row; i++) {
198                 wx[i] = vector_mul(w, Train_set[i].data, Train_col);
199             }
200             double w_new[50];
201             double gradient_avg = 0;
202             for (int j = 0; j < Train_col; j++) {
203                 double gradient = 0;
204                 for (int i = 0; i < Train_row; i++) {
205                     double logistic = exp(wx[i]) / (1 + exp(wx[i])) - Train_set[i].value;
206                     gradient = gradient + logistic * Train_set[i].data[j];
207                 }
208                 w_new[j] = w[j] - alpha * (gradient + beta * w[j]);
209                 gradient_avg += abs(alpha * gradient);
210             }
```

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）

1.1. 采用训练集如下所示

	A	B	C
1	1	2	1
2	2	-1	0

采用测试集如下所示

	A	B	C
1	3	3	?

1.2. 首先在训练集每行样本前加 1，得到矩阵

$$\begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & -1 \end{bmatrix}$$

并初始化权重向量 w 为$(1 \quad 1 \quad 1)$，步长 $\mu = 1$

1.3. w 对训练集的每行样本加权求和得到

$$\mathrm{w} * x_A = 4$$

$$\mathrm{w} * x_B = 2$$

1.4. 更新每一维的 w 如下所示

$$\mathrm{w} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - 1 \begin{bmatrix} \left(\frac{e^4}{1+e^4} - 1\right) * 1 + \left(\frac{e^2}{1+e^2} - 0\right) * 1 \\ \left(\frac{e^4}{1+e^4} - 1\right) * 1 + \left(\frac{e^2}{1+e^2} - 0\right) * 2 \\ \left(\frac{e^4}{1+e^4} - 1\right) * 2 + \left(\frac{e^2}{1+e^2} - 0\right) * (-1) \end{bmatrix} = \begin{pmatrix} 0.137 \\ -0.744 \\ 1.917 \end{pmatrix}$$

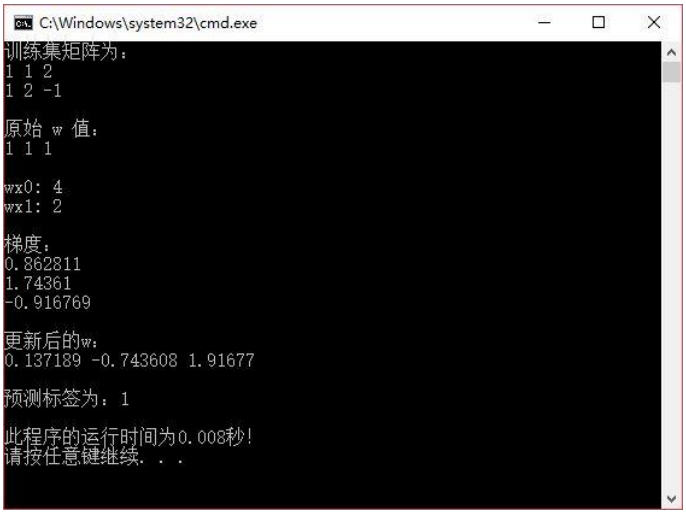
1.5. 用更新后的 w 对测试集进行测试

$$\mathrm{w} * x_{test} = 3.657$$

映射到 *logistic* 函数结果为

$$\frac{e^{3.657}}{1+e^{3.647}} = 0.975$$

1.6. 因为当前样本取值为 1 的概率大于 0.5，因此预测结果为 1。与程序运行结果预测一致。



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）

2.1. 最优结果（交叉验证法）

训练模型	准确率
随机梯度下降（动态步长）	75.4%

2.2. 优化结果分析（交叉验证法）

我们可以看到本次实验优化前后的准确率有略微的提高，但三种优化办法提高的效果都差不多，都比普通的逻辑回归模型提高了百分之两个点左右，**初步分析是此次数据集本身也是比较易分的，而且逻辑回归模型受参数影响较大**，在使用逻辑回归模型时，当我们调整学习率与正则化的一些参数时，对结果的影响比较巨大，从百分之五十多到百分之七十多的准确率的变化都有，所以我们可以看到其实逻辑回归模型就是为了通过合理的下降梯度寻找一个最佳的权重向量 w，**而不同的优化都只是为了更加快速更加方便的下降梯度得到最优的权重向量**，而调参也是如此，所以我们逻辑回归模型受参数的影响是巨大，参数调整好了也是可以得到一个最佳解。

注：由于这次实验数据集被随机打乱，且随机梯度下降也是概率事件，所以下面展示数据具有一定随机性。而且由于优化较多，展示时使用控制变量法，即只改变某优化看结果，所以只有同一个优化分析中的结果有可比性。

2.2.1. 基于最普通的逻辑回归（无任何优化）测试三种优化模型

训练模型	准确率
普通逻辑回归	73.45%
动态步长	75.3375%
随机梯度下降	75.4%
正则化	75.175%

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？

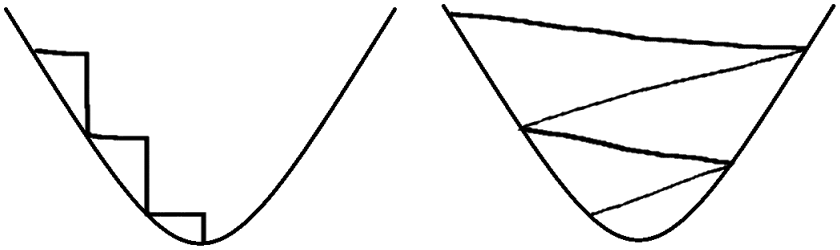
1.1. 由于梯度不是一个值，是一个矩阵，所以我们可能永远无法得到一个为 0 的梯度矩阵，如果我们将梯度为 0 作为算法停止的条件，我们可能会使算法陷入死循环，永远无法停止。而且如果我们的学习率设置的过大，最终的梯度可能在一个小范围内震荡，从而也无法收敛到 0。

1.2. 为了解决这个问题，我采取的办法是对梯度向量求一个平均值，当平均值小于某个我预先设好的阈值时则跳出迭代，哪怕如果一直无法达到这个阈值，我也设置了一个迭代上限，当迭代次数到达一定上限时算法也会自动停止，从而避免了陷入死循环的僵局。

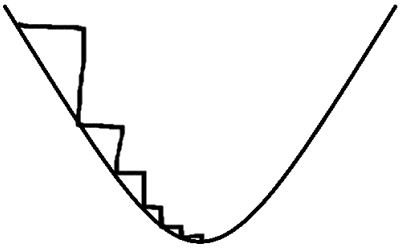
2. η 的大小会怎么影响梯度下降的结果？给出具体的解释，可视化的解释最好，比如图形展示等。

2.1. 学习率也叫学习步长，计算梯度是找到了更新 w 的方向，往这个方向更新的幅度则由 η 确定，η 的设置会直接影响迭代解能否求解到梯度的全局最优值。

2.2. 如左图所示，当我们的学习率 η 较小时，随着迭代次数的增加，w 便能随着梯度缓慢下降，最终下降到最低点，从而得到最优解。而如果我们的学习率 η 较大便可能出现右图的情况，w 不收敛反而发散，这是由于w是一个凸函数的，如果步长过大则会导致w反而随着梯度往增加的方向不断震荡，从而发散出去。在本次实验中便由于一开始的学习率过大，导致w的值过大，在计算e的次方时爆内存导致出错。



2.3. 当然如果我们的学习率设置过小，虽然不会导致发散的情况，但会导致训练时间过长，需要极大的迭代次数才能收敛到最优解，所以我们需要设置一个合理的学习率来进行梯度下降。我们还有另一个选择便是动态调整步长，如下图所示，我们可以随着迭代次数的增加，越接近最优解时则越减少步长，这样可以较快而且更加准确的迭代到最优解。



3. 思考这两种优化方法的优缺点：批梯度下降、随机梯度下降

3.1. 批梯度下降的时候，每次更新 w 时都需要遍历整个数据集，因此迭代速率较慢，特别是对于大数据集而言，计算复杂度过高，但是当学习率设置合理时，收敛所需的迭代次数相对于随机梯度下降是较少的，也不会出现明显的震荡现象。因此批梯度下降的时候，可通过判断梯度是否减小，即更新后的 w 是否小于更新前的 w 确定。

3.2. 随机梯度下降算法，每次迭代只是考虑让该样本点的 w 趋向最小，每次仅用一行样本来更新 w 而不管其他的样本点，此时迭代速率大大加快，这样算法会很快，但是收敛的过程会比较曲折，w 是震荡收敛的，因此需要将迭代次数设为较大，整体效果上，大多数时候它只能接近局部最优解，而无法真正达到局部最优解，所以适合用于较大训练集的数据。而且随机梯度下降的时候，每一步更新 w 时，无需将梯度是否减小作为更新依据，因为梯度是震荡减小的，只要将步长设置为较小的值，迭代较大次数，便可收敛。