哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

机器学习实验报告

课程类型 机器学习

题　　目 logistic 回归

学　　号 1160300507

姓　　名 聂晨晞

计算机科学与技术学院

目录

第 1 章 实验目的

理解逻辑回归模型，掌握逻辑回归模型的参数估计算法。

第 2 章 实验要求及实验环境

2.1 实验要求

1. 实现两种损失函数的参数估计（1，无惩罚项；2. 加入对参数的惩罚）
2. 采用梯度上升法求解 logistic 回归
3. 采用牛顿法求解 logistic 回归
4. 通过自己产生的数据评价该模型
5. 对当数据不符合朴素贝叶斯假定的时候模型结果进行讨论
6. 对从 uci 上下载的数据进行求解
7. 语言不限，可以用 matlab,python。求解解析解是可以利用现成的矩阵求逆。梯度下降，共轭梯度要求自己求梯度，迭代优化自己写。不允许利用现有的平台例如 pytorch，tensorflow 的自动微分工具

2.2 实验环境

1. 硬件环境：macbook pro mid-2017，3.1Ghz i5 处理器，16GB 内存
2. 软件环境：macOS 10.14+pycharm professional 2018.1+python 版本：3.6.4

第 3 章 设计思想

3.1 数学符号的定义

在讨论各个具体算法之前，本文定义如下的若干符号

1. m 为训练集中样本的个数
2. (n + 1) 为每一个样本的维度（在每一条数据的首部都添上常数项 1）
3. logistic 回归中的两类数据分别用数据 0 和数据 1 表示
4. $\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$ 表示有这 m 条（n+1）维向量组成的数据矩阵

5. $\boldsymbol{x_i} = \begin{bmatrix} 1 & x_{i1} & x_{i2} & \cdots & x_{in} \end{bmatrix}^T$ 表示 X 中的第 i 个数据
6. $\boldsymbol{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}^T$ 其中 $y_i \in \{0, 1\}$ 表示第 i 个数据属于哪一类
7. $\boldsymbol{h} = \begin{bmatrix} sigmoid(x_1) & sigmoid(x_2) & \cdots & sigmoid(x_m) \end{bmatrix}^T$ 表示由 logistic 回归中 sigmoid 函数给出的输出向量，对于每一个 $sigmoid(x_i)$ 如果 $sigmoid(x_i) < 0.5$ 在预测时就将其归为第 0 类，如果 $sigmoid(x_i) > 0.5$ 就将其归为第 1 类
8. $\boldsymbol{w} = \begin{bmatrix} w_0 & w_1 & \cdots & w_n \end{bmatrix}^T$ 表示对于 $\boldsymbol{X}$ 中的每一个 $(n+1)$ 维的数据，它在计算 sigmoid 函数的时候前面的系数是多少。

3.2 使用梯度上升法求解 logistic 回归

3.2.1 算法原理（1）——似然和 cost 函数

对于一个 0/1 分类问题，我们定义 sigmoid 函数如下所示：

$$h(\boldsymbol{x_i}) = g(w^T x_i) \frac{1}{1+e^{-w^T x_i}} \qquad sigmoid \tag{1}$$

这个函数的图像如图3.1所示

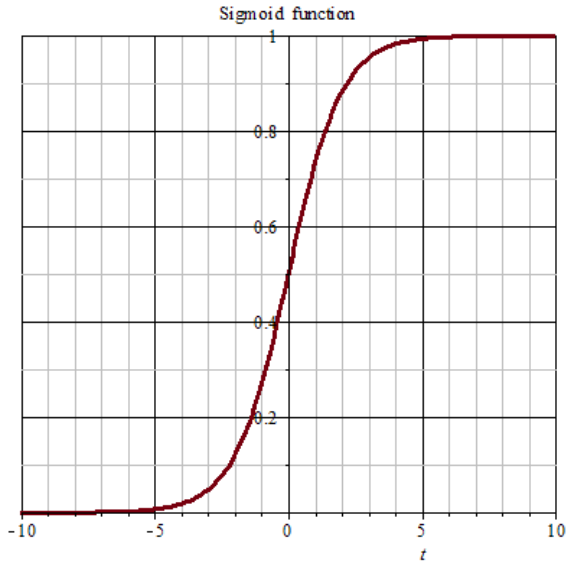


图 3.1: sigmoid 函数图像

从图中可以看出，该函数在 0～1 之间并且在 0.5 处进行分界，若 x>0.5 那么该函数迅速接近 1，否则迅速接近 0。于是我们很自然的可以将这个函数作为我们的预测函数，于是我们可以将预测的概率定义为如下2形式

$$\begin{aligned} P(Y = 1|\boldsymbol{x_i}; \boldsymbol{w}) &= h(\boldsymbol{x_i}) \\ P(Y = 0|\boldsymbol{x_i}; \boldsymbol{w}) &= 1 - h(\boldsymbol{x_i}) \end{aligned} \tag{2}$$

综合起来就是

$$P(Y|\boldsymbol{x_i};\boldsymbol{w}) = h(\boldsymbol{x_i})^y(1-h(\boldsymbol{x_i}))^{(1-y)} \tag{3}$$

于是对于所有的 m 组数据，我们可以将它们出现预测值的概率，在假设他们相互独立的情况下，写成如下的形式

$$\begin{aligned} P(Y|\boldsymbol{X};\boldsymbol{w}) &= \prod_{i\in\{1,m\}} P(Y|\boldsymbol{x_i};\boldsymbol{w}) \\ &= \prod_{i\in\{1,m\}} h(\boldsymbol{x_i})_i^y(1-h(\boldsymbol{x_i}))^{(1-y_i)} \end{aligned} \tag{4}$$

于是我们需要通过训练集学习出一个 $\boldsymbol{w}$ 使得等式4达到最大值。为了计算方便我们对等式4两边求对数得到如下的结果

$$J(w) = \log(P(Y|\boldsymbol{X};\boldsymbol{w})) = \sum_{i=1}^{m} y_i \log h(\boldsymbol{x_i}) + (1-y_i \log(1-h(x_i)) \tag{5}$$

接下来的任务就是想办法让公式5最大。相对应的，cost 函数可以用如下的公式表示

$$cost(x_i) = \begin{cases} -\log(h(\boldsymbol{x_i})) & y=1 \\ -\log(1-h(\boldsymbol{x_i})) & y=0 \end{cases} \tag{6}$$

3.2.2 算法原理 (2)——使用梯度上升法求解 likelihood 的最大值

在得到公式5之后，我们所需要做的就只是求出它的最大值，这可以通过上一个实验中提到的梯度下降法的变种梯度上升法来解决。该方法的核心思想是：如果要找某一个凹函数的最大值，我们应该沿着梯度的方向探寻，如果梯度记为 ∇ 那么，函数 $f(x,y)$ 的梯度是

$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix} \tag{7}$$

在梯度上升法中，梯度算子沿着函数增长最快的方向移动，记每一次移动的大小为 α（步长），那么梯度上升法的迭代公式为:

$$w = w + \alpha \nabla_w f(w) \tag{8}$$

我们将上一节中总结出来的公式5代入上式，问题就被转换成为:

$$w = w + \alpha \frac{\partial J(w)}{\partial w} (j = 1, 2, 3...n) \tag{9}$$

于是我们对于公式5求偏倒，可以得到如下的结果

$$\begin{aligned}
\frac{\partial}{\partial w_j}J(w) &= \sum_{i=1}^{m}(y^{(i)}\frac{1}{h_w(x^{(i)})}\frac{\partial h_w(x^{(i)})}{\partial w_j} + (1-y^{(i)})\frac{1}{1+h_w(x^{(i)})}\frac{\partial h_w(x^{(i)})}{\partial w_j}) \\
&= \sum_{i=1}^{m}(y^{(i)}\frac{1}{h_w(x^{(i)})} + (1-y^{(i)})\frac{1}{1+h_w(x^{(i)})})\frac{\partial h_w(x^{(i)})}{\partial w_j} \\
&= \sum_{i=1}^{m}(y^{(i)}\frac{1}{h_w(x^{(i)})} + (1-y^{(i)})\frac{1}{1+h_w(x^{(i)})})\frac{1}{(1+e^{-w^Tx})^2}e^{-w^Tx}\frac{\partial z(x)}{\partial w_j} \\
&= \sum_{i=1}^{m}(y^{(i)}\frac{1}{h_w(x^{(i)})} + (1-y^{(i)})\frac{1}{1+h_w(x^{(i)})})\frac{1}{(1+e^{-w^Tx})}\frac{e^{-w^Tx}}{(1+e^{-w^Tx})}\frac{\partial z(x)}{\partial w_j} \\
&= \sum_{i=1}^{m}(y^{(i)}\frac{1}{h_w(x^{(i)})} + (1-y^{(i)})\frac{1}{1+h_w(x^{(i)})})h_w(x^{(i)})(1+h_w(x^{(i)}))\frac{\partial w^Tx}{\partial w_j} \\
&= \sum_{i=1}^{m}(y^{(i)}(1+h_w(x^{(i)})) + (1-y^{(i)})h_w(x^{(i)}))x_j^{(i)} \\
&= \sum_{i=1}^{m}(y^{(i)} - h_w(x^{(i)}))x_j^{(i)}
\end{aligned} \tag{10}$$

于是我们的梯度上升算法，可以更新为如下的形式

$$w_j = w_j + \alpha\sum_{i=1}^{m}(y^{(i)} - h_w(x^{(i)}))x_j^{(i)} \tag{11}$$

同样的，为了加快计算过程，我们可以对这个梯度上生算法进行矩阵化，得到相应的矩阵化的算法如下所示：

$$w_{new} = w_{old} + \alpha X^T[\boldsymbol{y} - h(X^Tw)] \tag{12}$$

至此，我们完成了对梯度上升算法的讨论

3.2.3 算法实现

本次实验中，在公式12的指导下，可以直接通过 numpy 的矩阵表示进行相应的计算工作。

```
for i in range(max_iter):
    h = self.sigmoid(self.__data.dot(self.__w)) # m * 1
    self.__w = self.__w + alpha * self.__X.transpose().dot(self.__y - h)
```

3.3 使用梯度上升算法求解带有正则项的 logistic 回归模型

3.3.1 算法原理

使用梯度上升算法解决带有正则项的 logistic 回归模型和上一节中讨论的不带有正则项的 logistic 回归模型非常相似，也和上一个实验中的带有正则项的多项式拟合非常接近。这里不再重复推导，直接给出矩阵化之后的算法结果如下所示

$$w_{new} = w_{old} + \alpha\eta w_{old} + \alpha X^T[\boldsymbol{y} - h(X^T w)] \tag{13}$$

3.3.2 算法实现

本算法的实现和上一个不带正则项的算法非常接近，只是在更新 w 的时候加入了相应的惩罚项而已

```
for i in range(max_iter):
    h = self.sigmoid(self.__data.dot(self.__w)) # m * 1
    self.__w = self.__w + alpha * eta * self.__w \\
           +alpha * self.__X.transpose().dot(self.__y - h)
```

3.4 使用牛顿法求解 logistic 回归

3.4.1 算法原理（1）——牛顿法概述

牛顿法是求解方程 $f(x) = 0$ 的一种十分方便的解法，首先找到一个初始值 x_0 如果这个 x_0 不是方程的解，那么，做经过这个点的切线，该切线和 x 轴的交点为 x_1，同样，如果 x_1 不是该方程的解，我们继续做经过 $(x_1, f(x_1))$ 的切线，和 x 轴的交点为 x_2 以此类推，我们的 x_i 就会越来越逼近 $f(x) = 0$ 的解。

判断 x_i 是不是该方程的解有两种方法，一种是可以直接将 x_i 带入方程进行验证，也可以计算两次 x_i 之间的距离，如果该距离少于某一个阈值，我们就判断牛顿法收敛。

经过的某一点 $(x_i, f(x_i))$ 的切线为 $f(x) = f(x_i) + f'(x_i)(x - x_i)$ 注意，这条直线同样也是泰勒展开的一次形式（省略极小项），于是我们让该切线方程等于 0，得到如下的式子

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{14}$$

由上式可以看出，我们实际上得到了一个迭代公式，即通过一个已有的 x_i 以及 x_i 的其他已知信息求出下一个 x_{i+1} 的方法

3.4.2 算法原理（2）——牛顿法在优化问题中的应用

对于某一个待优化的函数 f，通过以往的数学知识我们知道这个函数的极值往往出现在 $f' = 0$ 的点上，于是我们可以将解决函数 l 的优化问题转化为一个解方程的问题，进而使用牛顿法进行求解。通过一个相似的过程，我们很容易得到此时的迭代公式为

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)} \tag{15}$$

但是在实际的应用中，我们的函数 X 往往不是一个单变量，而是一个向量，于是我们需要将牛顿法从现在讨论的单一变量的 x 扩展为一个向量 $\boldsymbol{X}$ 的函数，过程如下所示：

对于某一个向量的函数 $f(\boldsymbol{X})$ 来说，我们可以将其在 $\boldsymbol{X_0}$ 处进行二阶泰勒展开，得到如下的结果：

$$f(\mathbf{X}) = f(\mathbf{X}_0) + (\mathbf{X} - \mathbf{X}_0)^T \nabla f(\mathbf{X}_0) + \frac{1}{2}(\mathbf{X} - \mathbf{X}_0)^T \mathbf{H} f(\mathbf{X}_0)(\mathbf{X} - \mathbf{X}_0) + o(\|\mathbf{X} - \mathbf{X}_0\|^2) \quad (16)$$

其中 $o(\|\mathbf{X} - \mathbf{X}_0\|^2)$ 是一个高阶无穷小，表示皮阿诺余项，而 $\mathbf{H}f(\mathbf{X}_0)$ 是一个 Hassian 矩阵，我们在 ΔX 很小的时候，可以忽略皮阿诺余项，得到如公式**??**

$$\mathbf{X}_{n+1} = \mathbf{X}_n - \frac{\nabla f(\mathbf{X}_n)}{\mathbf{H} f(\mathbf{X}_n)}, \quad n \geq 0 \quad (17)$$

由此我们将牛顿法从一个一维的函数扩展为了一个多维向量的形式

3.4.3 算法原理（3）——牛顿法求解 logistic 回归

对于 logistic 回归的问题，我们对 likelihood 求导得到了公式10结果

$$\frac{\partial}{\partial w_j} J(w) = \sum_{i=1}^{m} (y^{(i)} - h_w(x^{(i)})) x_j^{(i)} \quad (18)$$

由于 $J(w)$ 是一个多元函数，那么我们对于所有的 $w_0, w_1, w_2, \cdots, w_n$ 应该都有对应的偏导数为 0，于是我们连立如下的方程组

$$\begin{aligned}
\frac{\partial}{\partial w_0} J(w) &= \sum_{i=1}^{m} (y^{(i)} - h_w(x^{(i)})) x_0^{(i)} \\
\frac{\partial}{\partial w_1} J(w) &= \sum_{i=1}^{m} (y^{(i)} - h_w(x^{(i)})) x_1^{(i)} \\
\frac{\partial}{\partial w_2} J(w) &= \sum_{i=1}^{m} (y^{(i)} - h_w(x^{(i)})) x_2^{(i)} \\
&\vdots \\
\frac{\partial}{\partial w_n} J(w) &= \sum_{i=1}^{m} (y^{(i)} - h_w(x^{(i)})) x_n^{(i)}
\end{aligned} \quad (19)$$

在求解这个方程组之前，我们需要证明似然函数是一个凹函数，我们可以通过证明该似然函数的 Hassian 矩阵是一个负定阵来证明，过程如下：

在求 Hassian 矩阵之前的我们需要得到似然函数的二阶导，

$$\begin{aligned}
\frac{\partial^2 \ln L(\mathbf{w})}{\partial w_k \partial w_j} &= \frac{\partial \sum_{i=1}^{m} x_{ik}[y_i - \pi(\mathbf{x}_i)]}{\partial w_j} \\
&= \sum_{i=1}^{m} x_{ik} \frac{\partial(-\frac{e^{\mathbf{x}_i}}{1+e^{\mathbf{x}_i}})}{\partial w_j} \\
&= -\sum_{i=1}^{m} x_{ik} \frac{\partial(1+e^{-\mathbf{x}_i})^{-1}}{\partial w_j} \\
&= -\sum_{i=1}^{m} x_{ik}[-(1+e^{-\mathbf{x}_i})^{-2} \cdot e^{-\mathbf{x}_i} \cdot (-\frac{\partial \mathbf{x}_i}{\partial w_j})] \\
&= -\sum_{i=1}^{m} x_{ik}[(1+e^{-\mathbf{x}_i})^{-2} \cdot e^{-\mathbf{x}_i} \cdot x_{ij}] \\
&= -\sum_{i=1}^{m} x_{ik} \cdot x_{ij} \cdot \frac{1}{1+e^{\mathbf{x}_i}} \cdot \frac{e^{\mathbf{x}_i}}{1+e^{\mathbf{x}_i}} \\
&= \sum_{i=1}^{m} x_{ik} \cdot x_{ij} \cdot \pi(\mathbf{x}_i) \cdot [\pi(\mathbf{x}_i) - 1] \\
&= \sum_{i=1}^{m} x_{ik} \cdot \pi(\mathbf{x}_i) \cdot [\pi(\mathbf{x}_i) - 1] \cdot x_{ij}
\end{aligned} \tag{20}$$

进而我们通过 Hassian 矩阵来表示二阶导有：

$$\mathbf{X} = \begin{bmatrix} x_{10} & x_{11} & \cdots & x_{1n} \\ x_{20} & x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m0} & x_{m1} & \cdots & x_{mn} \end{bmatrix} \tag{21}$$

$$\mathbf{A} = \begin{bmatrix} \pi(\mathbf{x}_1) \cdot [\pi(\mathbf{x}_1) - 1] & 0 & \cdots & 0 \\ 0 & \pi(\mathbf{x}_2) \cdot [\pi(\mathbf{x}_2) - 1] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \pi(\mathbf{x}_m) \cdot [\pi(\mathbf{x}_m) - 1] \end{bmatrix} \tag{22}$$

$$\boldsymbol{H} = X^T \boldsymbol{A} X \tag{23}$$

显然，A 是负定的，那么根据线性代数的知识，我们可以知道，似然函数的 Hassian 矩阵也是负定的

对于多元函数的零点求解同样可以通过牛顿法求解，对于当前讨论的 logistic 回归，我们有如下迭代式：

$$\mathbf{X}_{new} = \mathbf{X}_{old} - \frac{\mathbf{U}}{\mathbf{H}} = \mathbf{X}_{old} - \mathbf{H}^{-1}\mathbf{U} \tag{24}$$

其中，

$$\mathbf{U} = \begin{bmatrix} x_{10} & x_{20} & \cdots & x_{m0} \\ x_{11} & x_{21} & \cdots & x_{m1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1n} & x_{2n} & \cdots & x_{mn} \end{bmatrix} \begin{bmatrix} y_1 - \pi(\mathbf{x}_1) \\ y_2 - \pi(\mathbf{x}_2) \\ \vdots \\ y_m - \pi(\mathbf{x}_m) \end{bmatrix} \tag{25}$$

至此我们完成了对牛顿法解决 logistic 回归的讨论，下一节给出关键的实现代码。

3.4.4 算法实现

牛顿法求解 logistic 回归的算法通过 python 中 numpy 的矩阵操作实现起来非常方便，主要代码如下所示

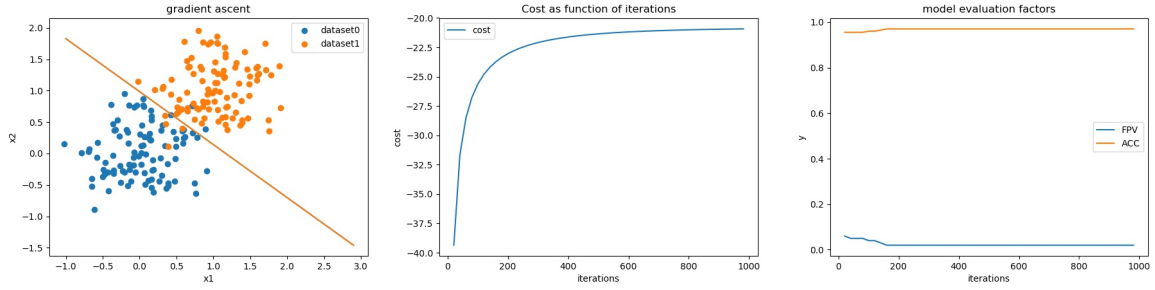
```
h = self.sigmoid(self.__data.dot(self.__w)) # m * 1
A = np.diag([float(h[i] * (1 - h[i])) for i in range(len(h))])
H = self.__data.transpose().dot(A).dot(self.__data)
error = self.__label - h
U = self.__data.transpose().dot(error)
self.__w = self.__w + inv(H).dot(U)
```

第 4 章 实验结果与分析

4.1 使用梯度上升法求解 logistic 回归

4.1.1 在符合朴素贝叶斯假设的情况下的求解

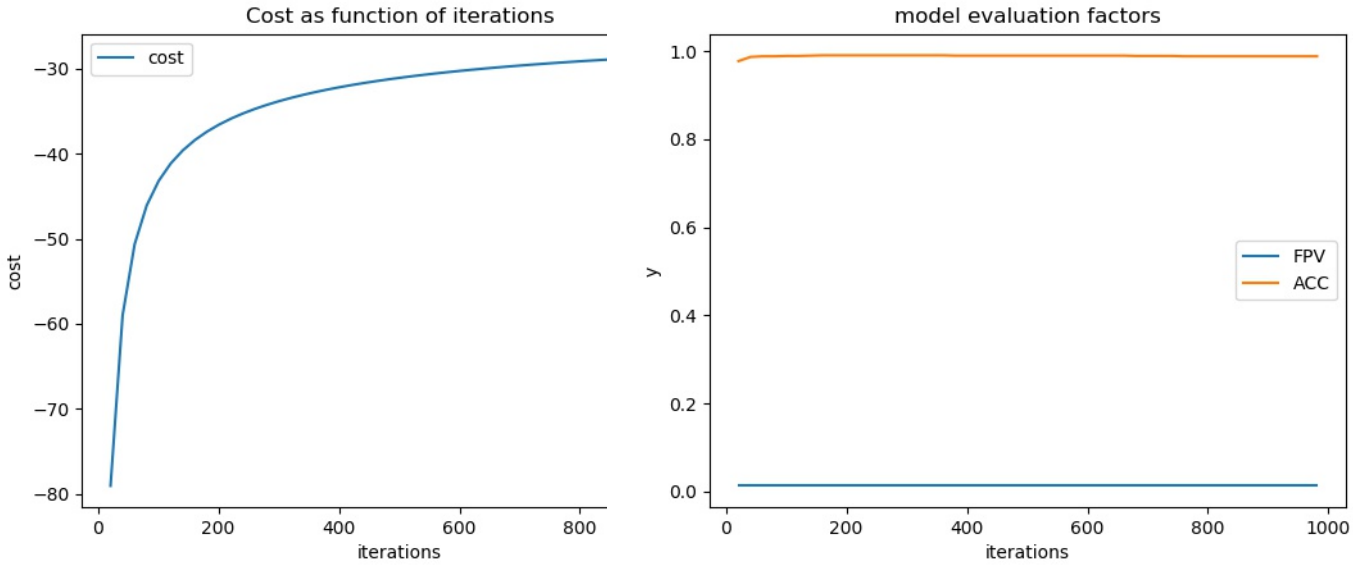
对于梯度上升法求解 logistic 回归，在给定步长为 0.01，训练 600 次时，将所有的数据的 cost 函数的函数值加起来之后可以得到一个总的 cost 值是-20.9，训练得到的最后的结果以及训练过程中，cost 函数，accuracy，以及 FPV 的曲线如图所示



(a) 求解生成数据最终结果　(b) 生成数据上 cost 之和随迭代次数变化　(c) 生成数据上 FPV,accuracy 随迭代次数变化

图 4.1: 梯度上升法求解在生成数据上的 logistic 回归

由于 uci 上下载的数据是四维的数据，这个数据并不可以进行可视化，于是在使用梯度上升的时候，它输出的 cost 之和和 accuracy，FPV 曲线如下所示



(a) uci 数据上 cost 之和随着迭代次数变化　　(b) uci 数据上 FPV 和 accuracy 随着迭代次数的变化

图 4.2: 梯度上升求解在 uci 数据上的 logistic 回归

在不符合朴素贝叶斯假设的情况下进行求解

当产生数据并不符合朴素贝叶斯假设的时候，进行试验求解，对于两组数据的方差不同的情况，我分别进行了在原来基础上调小和在原来基础上调大的两次实验，得到的结果分别如图4.3所示。

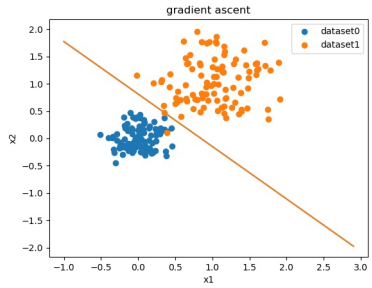
从以上讨论中我们可以得到结论，如果两组数据的方差并不一样，我们的模型仍然能给出相对正确的结果，但是如果二者的方差太大以至于二者相互混合在一起的程度很高，那么我们的模型也无法得到正确的结果

4.2 使用牛顿法求解

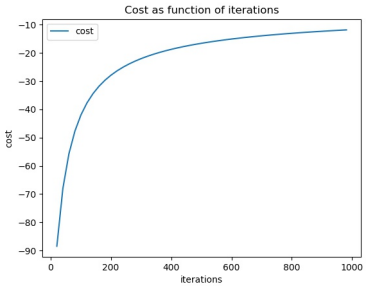
使用牛顿法对相同的一组数据进行求解的一个显著的特点就是循环的次数显著减小，为了达到大体相同的训练效果，梯度上升算法在步长设置为 0.01 的时候可能需要大约 500 次的迭代，但是如果使用牛顿法，仅仅需要 5 次左右。

图4.4的结果是通过牛顿法在设置训练次数为 8 次的时候得到的结果。从结果中可以看出在 8 次训练的情况下牛顿法在生成数据上给出了相当好的分类结果

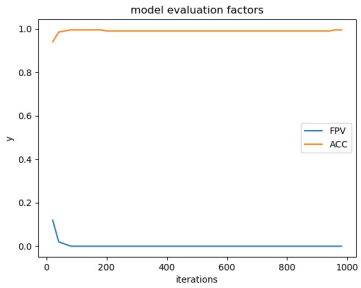
同样，我们将牛顿法在 uci 的数据中进行了测试，得到的结果如图4.5所示



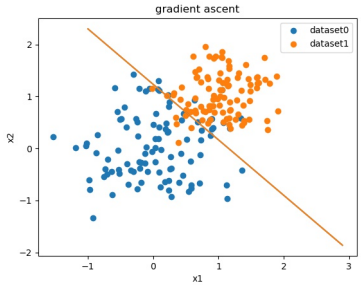
(a) 方差减小时的结果



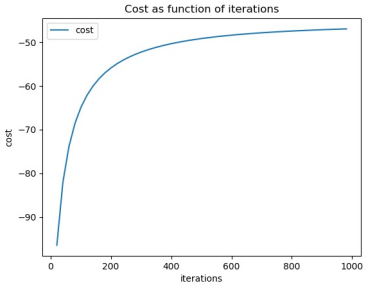
(b) 方差减小时 cost 函数



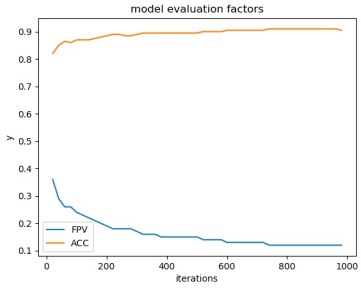
(c) 方差减小是 FPV 和 accuracy 函数



(d) 方差增加时的结果

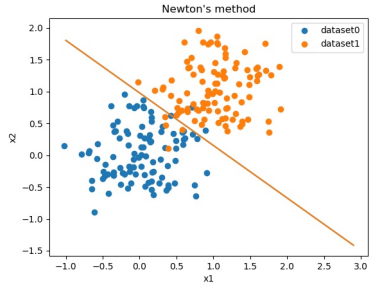


(e) 方差增加时 cost 函数

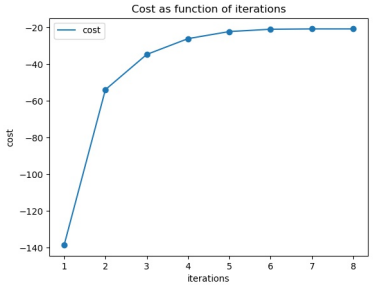


(f) 方差增加是 FPV 和 accuracy 函数

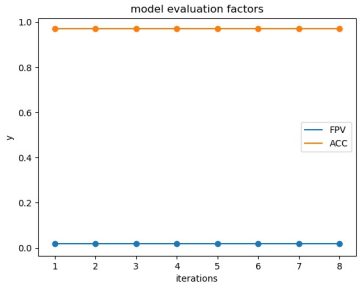
图 4.3: 方差增加和减少时对 logsitic 回归的求解



(a) 牛顿法训练结果

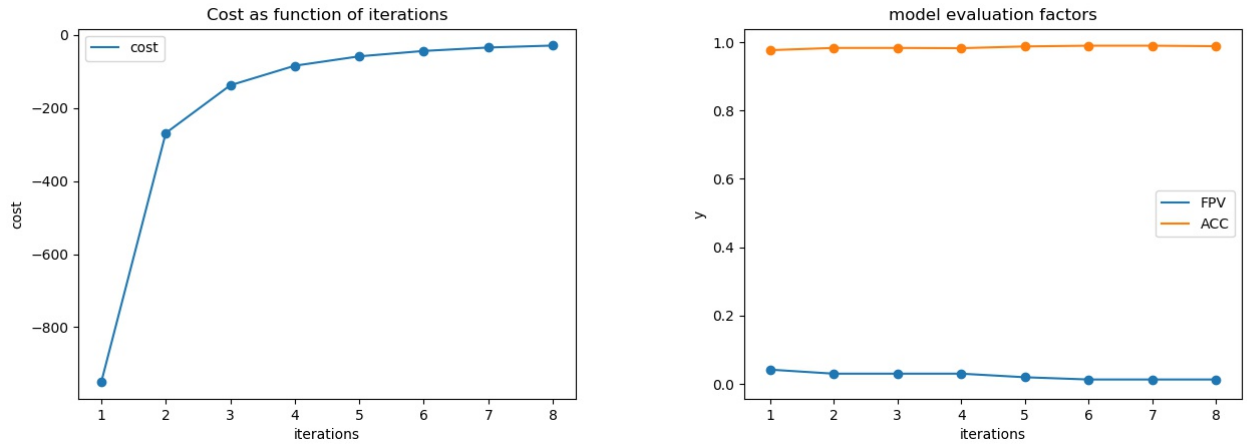


(b) 牛顿法 cost 函数



(c) 牛顿法 FPV 和 accuracy

图 4.4: 牛顿法在生成数据上得到的结果



(a) 牛顿法 cost 函数

(b) 牛顿法 FPV 和 accuracy

图 4.5: 牛顿法在 uci 数据上的结果

第 5 章 结论

本文中，我们一共尝试了两种求解 logistic 回归的方法，一为梯度上升法，一为牛顿法，其中牛顿法在初值选择恰当的情况下对比梯度上升法有着明显的优势，可以显著降低训练的次数，但是牛顿法由于需要产生 Hassian 矩阵，在存储的要求上较梯度上升法高。

第 6 章 参考文献

'Pattern recognition and machine learning' Christopher M. Bishop
牛顿法英文维基百科 https://en.wikipedia.org/wiki/Newton%27s_method
梯度下降法英文维基百科 https://en.wikipedia.org/wiki/Gradient_descent
斯坦福机器学习课程讲义吴恩达 http://www.andrewng.org

A 源代码

本次实验和以后所有实验的源代码和实验报告都可以在这个 github 仓库中找到https://github.com/Billy-Nie/machine_learning_HIT