Linear and Logistic Regression-逻辑回归模型

王 成

上海交通大学数学科学学院

Section 1

分类背景知识

Classification

The problem of predicting a discrete random variable Y from another random variable X is called

- **classification**(分类);
- **supervised learning**(监督学习);
- **discrimination**(判别分析);
- **pattern recognition**(模式识别).

实例1-Iris Data Set

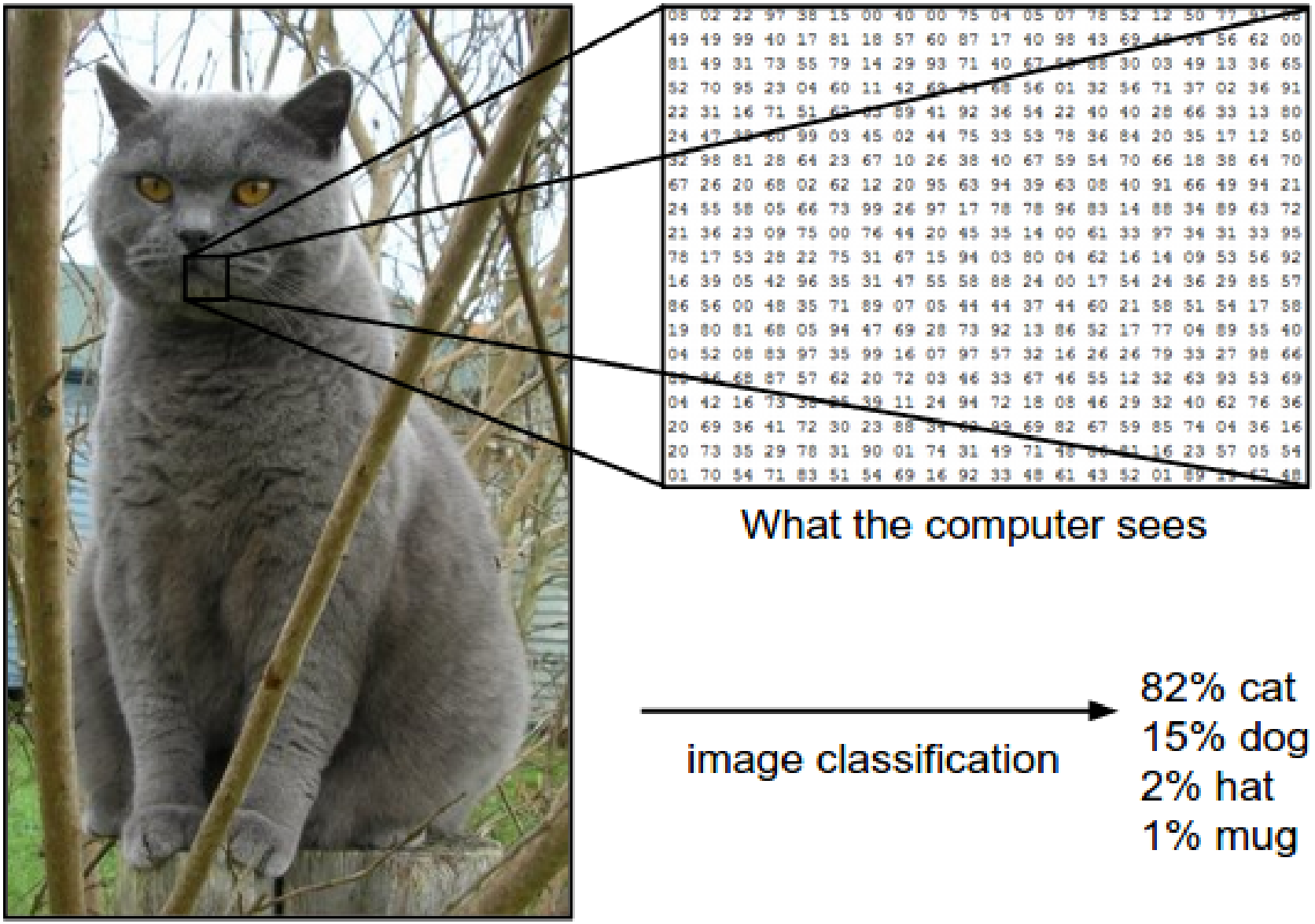
Iris Data Set(鸢尾属植物数据集)是历史最悠久的数据集，它首次出现在著名的英国统计学家和生物学家Ronald Fisher 1936年的论文中。在这个数据集中，包括了三类不同的鸢尾属植物：Setosa，Versicolour, Virginica。每类收集了50个样本，整个数据集一共包含了150个样本。该数据集测量了所有150个样本的4个特征，分别是：

- sepal length（花萼长度）
- sepal width（花萼宽度）
- petal length（花瓣长度）
- petal width（花瓣宽度）

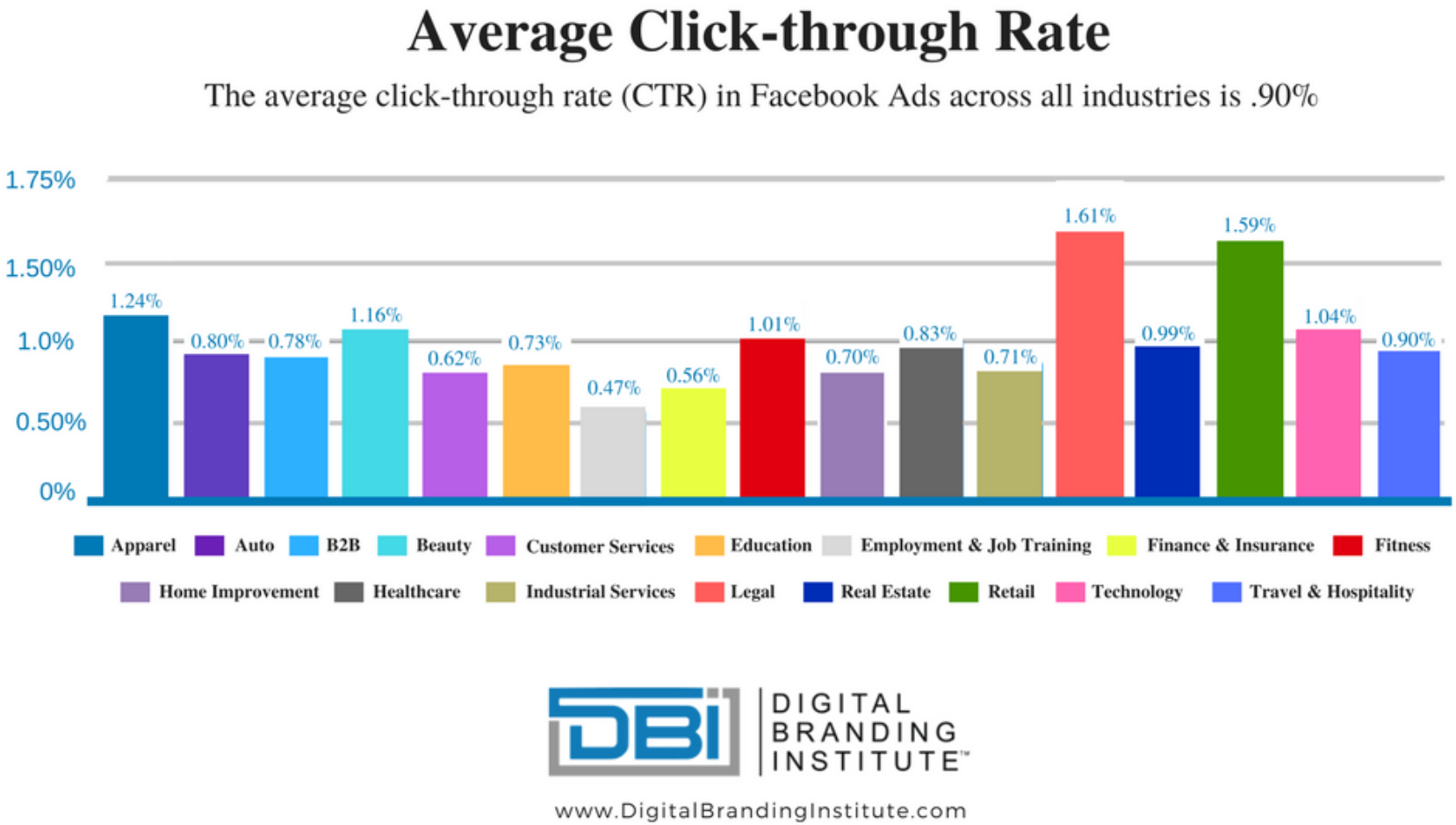
实例2-Spam email

The "spam" concept is diverse: advertisements for products/web sites, make money fast schemes, chain letters, pornography..
Our collection of spam e-mails came from our postmaster and individuals who had filed spam. Our collection of non-spam e-mails came from filed work and personal e-mails, and hence the word 'george' and the area code '650' are indicators of non-spam. These are useful when constructing a personalized spam filter. One would either have to blind such non-spam indicators or get a very wide collection of non-spam to generate a general purpose spam filter.

实例3-Visual Recognition



实例4-Click-Through Rate



Section 2

Logistic Regression-逻辑回归

逻辑回归

- 线性回归模型一般适用于被解释变量是连续的情形。
- 如果Y的取值本身是离散的，简单的线性回归模型就不再合适(因为其给出的预测不是离散的).

逻辑回归

- 线性回归模型一般适用于被解释变量是连续的情形。
- 如果Y的取值本身是离散的，简单的线性回归模型就不再合适(因为其给出的预测不是离散的).
- 我们如果还是使用最小二乘法，对应的统计方法是Fisher's LDA.

逻辑回归

- 线性回归模型一般适用于被解释变量是连续的情形。
- 如果Y的取值本身是离散的，简单的线性回归模型就不再合适(因为其给出的预测不是离散的).
- 我们如果还是使用最小二乘法，对应的统计方法是Fisher's LDA.

对于离散情形的应变量Y, 对应的是分类问题,最常用的就是逻辑回归和支持向量机.

错分率

对于分类问题，如果预测值也是0或者1，那么

$$\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2=\frac{1}{n}\sum_{i=1}^{n}I(y_i\neq\hat{y}_i),$$

恰好就是分类问题中错分的比例，即错分率。

线性回归做分类

对于每个β, 我们设定预测为$\hat{y}_i = I(\mathbf{X}_i^\mathsf{T}\beta > 0)$,

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq I(\mathbf{X}_i^\mathsf{T}\beta > 0)) = \frac{1}{n}\sum_{i=1}^{n} I((2y_i - 1)\mathbf{X}_i^\mathsf{T}\beta \leq 0),$$

这里$2y_i - 1$把$\{0, 1\}$转化为了$\{+1, -1\}$. 类似于最小二乘法，可以考虑

$$\hat{\beta} = \arg\min \frac{1}{n}\sum_{i=1}^{n} I((2y_i - 1)\mathbf{X}_i^\mathsf{T}\beta \leq 0).$$

线性回归做分类

对于每个β, 我们设定预测为$\hat{y}_i = I(\mathbf{X}_i^\top \beta > 0)$,

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq I(\mathbf{X}_i^\top \beta > 0)) = \frac{1}{n}\sum_{i=1}^{n} I((2y_i - 1)\mathbf{X}_i^\top \beta \leq 0),$$

这里$2y_i - 1$把$\{0, 1\}$转化为了$\{+1, -1\}$. 类似于最小二乘法，可以考虑

$$\hat{\beta} = \arg\min \frac{1}{n}\sum_{i=1}^{n} I((2y_i - 1)\mathbf{X}_i^\top \beta \leq 0).$$

缺点:

- 这里的损失函数不连续，优化问题很难求解.
- 优化问题可能具有多个解，不易解释. (统计性质也很难分析)

Subsection 1

逻辑回归原理

Logit Model

因为这里应变量y只取0或者1, 假设存在一个β,使得

$$r(\mathbf{x}) = P(Y = 1|X = \mathbf{x}) \propto \mathbf{x}^{\mathsf{T}}\beta.$$

思考

你能想到什么样的函数 $f: \mathbb{R} \to [0, 1]$?

Logit Model

定义一个连接函数(link function) $f : \mathbb{R} \to [0, 1]$,从解释性以及计算角度，还期待：

- 函数是单调增的;
- 函数是光滑连续的;
- 函数是常用的;
- $\cdots$

Logit Model

定义一个连接函数(link function) $f : \mathbb{R} \to [0, 1]$,从解释性以及计算角度，还期待：

- 函数是单调增的;
- 函数是光滑连续的;
- 函数是常用的;
- ...

逻辑回归采用的是Sigmoid函数:

$$r(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}^{\mathsf{T}}\beta}} = \frac{e^{\mathbf{x}^{\mathsf{T}}\beta}}{1 + e^{\mathbf{x}^{\mathsf{T}}\beta}}.$$

逻辑回归-极大似然估计

把y_i生成机制设定为概率为$r(\mathbf{X}_i)$的二项分布，写出似然函数

$$L(\beta) = \prod_{i=1}^{n} r(\mathbf{X}_i)^{y_i}(1 - r(\mathbf{X}_i))^{1-y_i} = \prod_{i=1}^{n} \frac{e^{y_i \mathbf{X}_i^{\mathsf{T}} \beta}}{1 + e^{\mathbf{X}_i^{\mathsf{T}} \beta}},$$

逻辑回归估计为

$$\begin{aligned}
\hat{\beta} &= \arg\max_{\beta} \prod_{i=1}^{n} \frac{e^{y_i \mathbf{X}_i^{\mathsf{T}} \beta}}{1 + e^{\mathbf{X}_i^{\mathsf{T}} \beta}} \\
&= \arg\max_{\beta} \frac{1}{n} \sum_{i=1}^{n} \{y_i \mathbf{X}_i^{\mathsf{T}} \beta - \log(1 + e^{\mathbf{X}_i^{\mathsf{T}} \beta})\} \\
&= \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \{\log(1 + e^{\mathbf{X}_i^{\mathsf{T}} \beta}) - y_i \mathbf{X}_i^{\mathsf{T}} \beta\}.
\end{aligned}$$

逻辑回归模型–Summary

给定样本$(\mathbf{X}_1, y_1), \cdots, (\mathbf{X}_n, y_n)$:

- Step 1: 对于任意一条线(任意一个β), 可以得到预测值:

$$\hat{y}_i = \frac{e^{\mathbf{X}_i^\mathsf{T}\beta}}{1 + e^{\mathbf{X}_i^\mathsf{T}\beta}} \in (0, 1).$$

- Step 2: 逻辑回归模型

$$\arg\min_{\beta \in \mathbb{R}^p} -\frac{1}{n}\sum_{i=1}^{n}\{y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\}.$$

文献中称Cross Entropy Loss(交叉熵损失).

Section 3

逻辑回归的算法

逻辑回归的算法

逻辑回归没有显示解，一般通过优化算法迭代的过程来完成。

几乎所有的统计方法都可以通过优化的形式表示成:

$$\hat{\beta} = \arg\min f(\beta).$$

逻辑回归的算法

逻辑回归没有显示解，一般通过优化算法迭代的过程来完成。

几乎所有的统计方法都可以通过优化的形式表示成:

$$\hat{\beta} = \arg\min f(\beta).$$

如果$f(\cdot)$是连续可导的,常用的优化算法

- Gradient descent (梯度下降);
- Newton's method (牛顿法)

Gradient descent

对于优化问题$\arg\min f(x)$, 给定一个初始点x_n, Gradient descent对函数做一个一阶Taylor展开 :

$$f(x) \approx f(x_n) + \nabla f(x_n)(x - x_n) + \frac{1}{2\gamma}\|x - x_n\|_2^2,$$

迭代过程为:

$$\begin{aligned} x_{n+1} =& \arg\min_x \{f(x_n) + \nabla f(x_n)(x - x_n) + \frac{1}{2\gamma}\|x - x_n\|_2^2\} \\ =& x_n - \gamma \nabla f(x_n). \end{aligned}$$

Newton's method

给定一个初始点x_n, Newton's method对函数做一个二阶Taylor展开：

$$f(x) \approx f(x_n) + \nabla f(x_n)(x - x_n) + \frac{1}{2}(x - x_n)^{\mathrm{T}} \nabla^2 f(x_n)(x - x_n),$$

迭代过程为:

$$\begin{aligned} x_{n+1} =& \arg\min_x \{f(x_n) + \nabla f(x_n)(x - x_n) + \frac{1}{2}(x - x_n)^{\mathrm{T}} \nabla^2 f(x_n)(x - x_n)\} \\ =& x_n - (\nabla^2 f(x_n))^{-1} \nabla f(x_n). \end{aligned}$$

Newton's method

给定一个初始点x_n, Newton's method对函数做一个二阶Taylor展开：

$$f(x) \approx f(x_n) + \nabla f(x_n)(x - x_n) + \frac{1}{2}(x - x_n)^{\mathrm{T}} \nabla^2 f(x_n)(x - x_n),$$

迭代过程为:

$$\begin{aligned} x_{n+1} =& \arg\min_x \{f(x_n) + \nabla f(x_n)(x - x_n) + \frac{1}{2}(x - x_n)^{\mathrm{T}} \nabla^2 f(x_n)(x - x_n)\} \\ =& x_n - (\nabla^2 f(x_n))^{-1} \nabla f(x_n). \end{aligned}$$

数学直观上，Newton法是二阶方法，梯度下降是一阶方法。前者有更好的逼近，算法收敛上会较快，缺点是需要求解二阶Hessian矩阵相关线性方程组，计算量上要大一点。

模拟数据

我们首先生成随机数据，其中

$$X_1, \cdots, X_n \ i.i.d. \ \sim N(0,1),$$
$$Y_i \sim B\left(1, \frac{e^{X_i}}{1+e^{X_i}}\right)$$

即首先生成n个来自于标准正态的解释变量X_i，然后选择真实逻辑回归系数$\beta_0 = 0, \beta_1 = 1$得到响应变量Y_i.

模拟数据

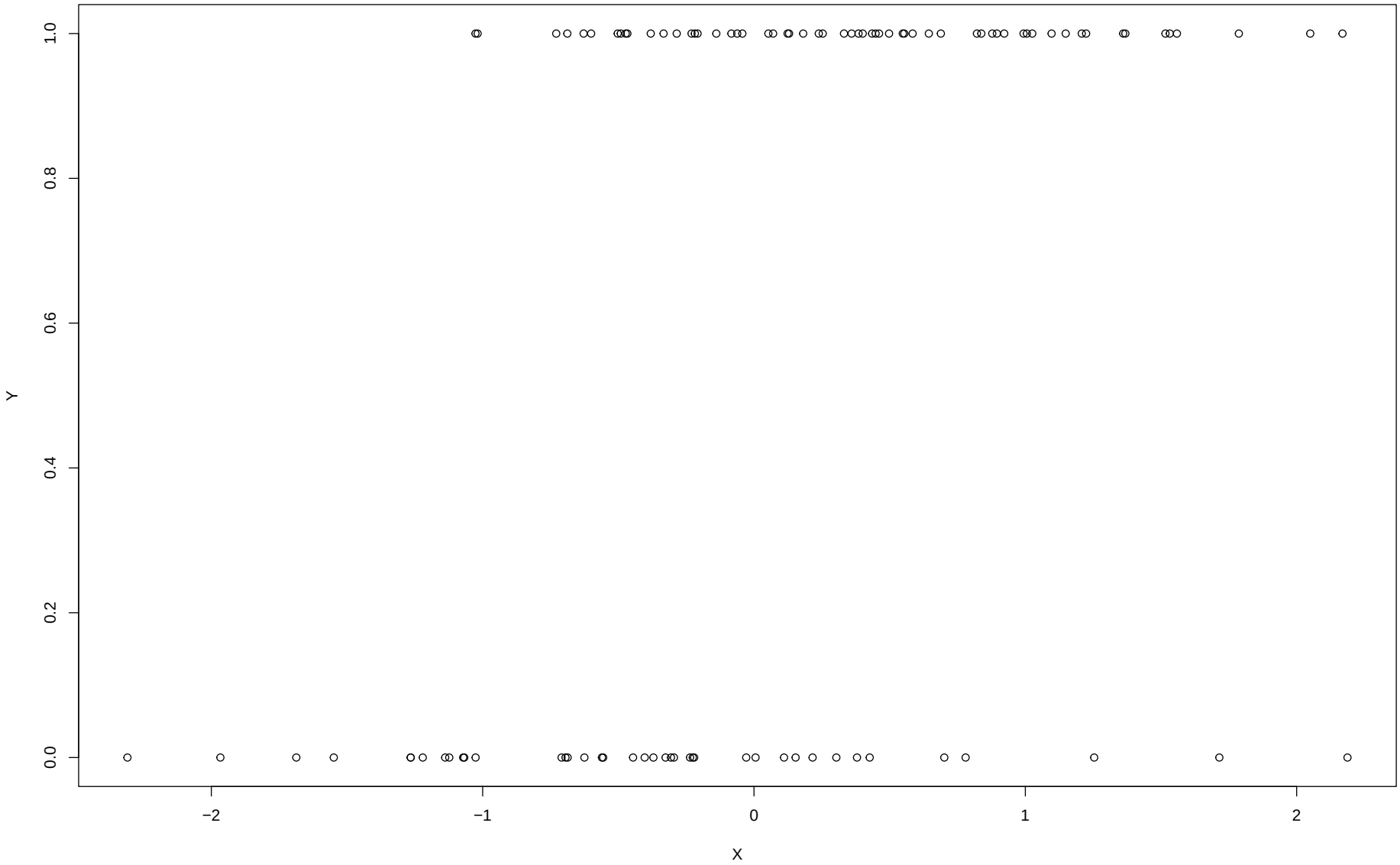
```
rm(list=ls()) ###初始化编程环境
set.seed(123) ##设置随机种子
n=100  ##设定样本大小n=100
X<-rnorm(n)   ##生成解释变量
Y=X   ##初始化响应变量
for (k in 1:n)
{
pr=exp(X[k])/(1+exp(X[k]))
Y[k]=rbinom(1,1,pr)
}
summary(X)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -2.30917 -0.49385  0.06176  0.09041  0.69182  2.18733
```

```
summary(Y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     1.0     0.6     1.0     1.0
```

模拟数据-Plot



逻辑回归优化问题

逻辑回归估计为

$$\begin{aligned}\hat{\beta} &= \arg\max_{\beta} \prod_{i=1}^{n} \frac{e^{y_i \mathbf{x}_i^{\mathsf{T}} \beta}}{1 + e^{\mathbf{x}_i^{\mathsf{T}} \beta}} \\ &= \arg\max_{\beta} \frac{1}{n} \sum_{i=1}^{n} \{y_i \mathbf{x}_i^{\mathsf{T}} \beta - \log(1 + e^{\mathbf{x}_i^{\mathsf{T}} \beta})\} \\ &= \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \{\log(1 + e^{\mathbf{x}_i^{\mathsf{T}} \beta}) - y_i \mathbf{x}_i^{\mathsf{T}} \beta\}.\end{aligned}$$

逻辑回归算法-Gradient descent

对于优化问题

$$\arg\min f(x),$$

Gradient descent的迭代过程为:

$$x_{n+1} = x_n - \gamma \nabla f(x_n),$$

其中γ是算法每一步的步长，$\nabla f(x)$为函数的偏导数，优化中称为梯度，记$x = (X_1, \ldots, X_p)'$

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial X_1} \\ \vdots \\ \frac{\partial f(x)}{\partial X_p}. \end{pmatrix}$$

逻辑回归算法-Gradient descent

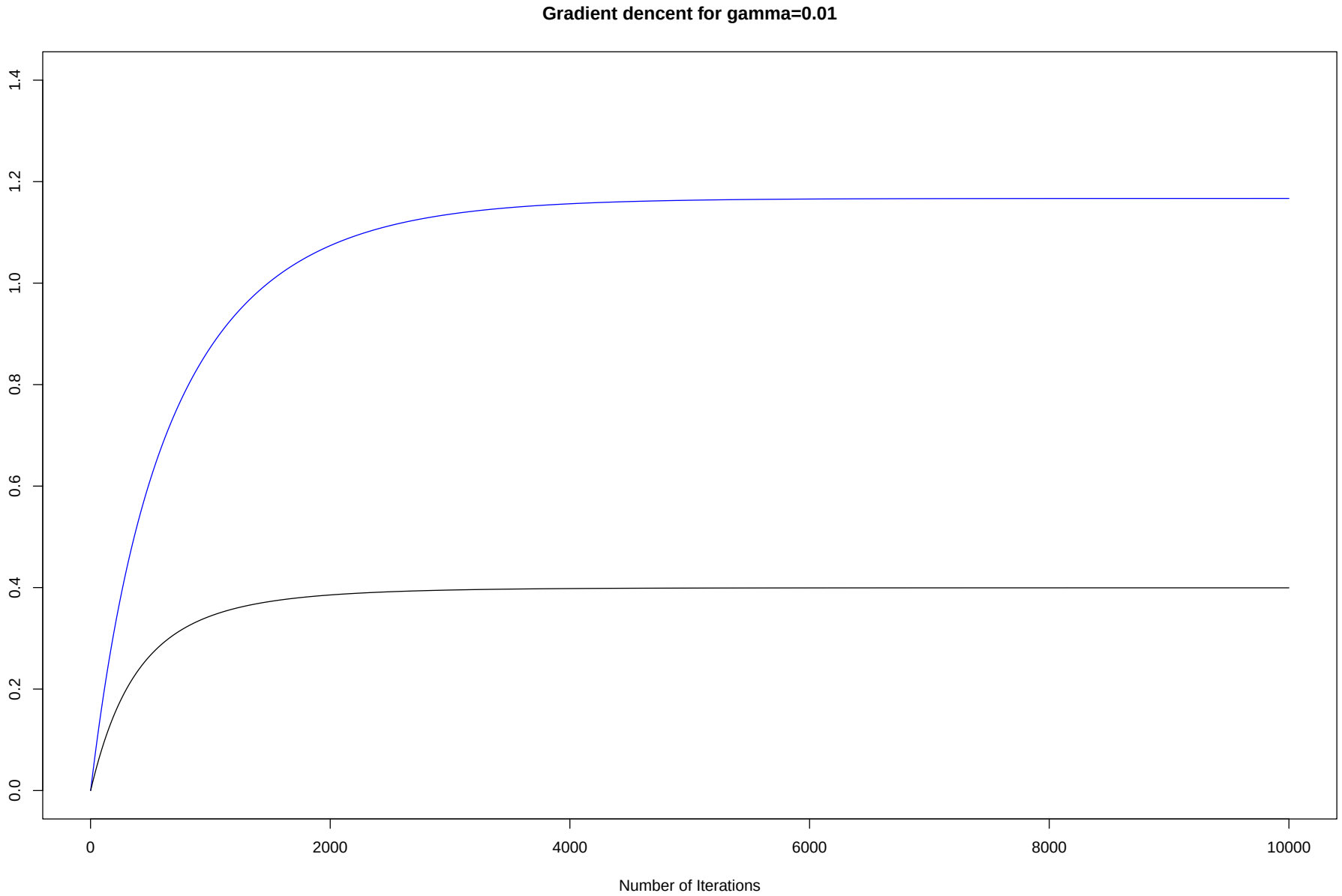
对于逻辑回归，我们可以得到梯度为

$$\frac{1}{n}\sum_{i=1}^{n}\{\frac{e^{\mathbf{X}_i^{\mathsf{T}}\beta}}{1+e^{\mathbf{X}_i^{\mathsf{T}}\beta}}-Y_i\}\mathbf{X}_i.$$

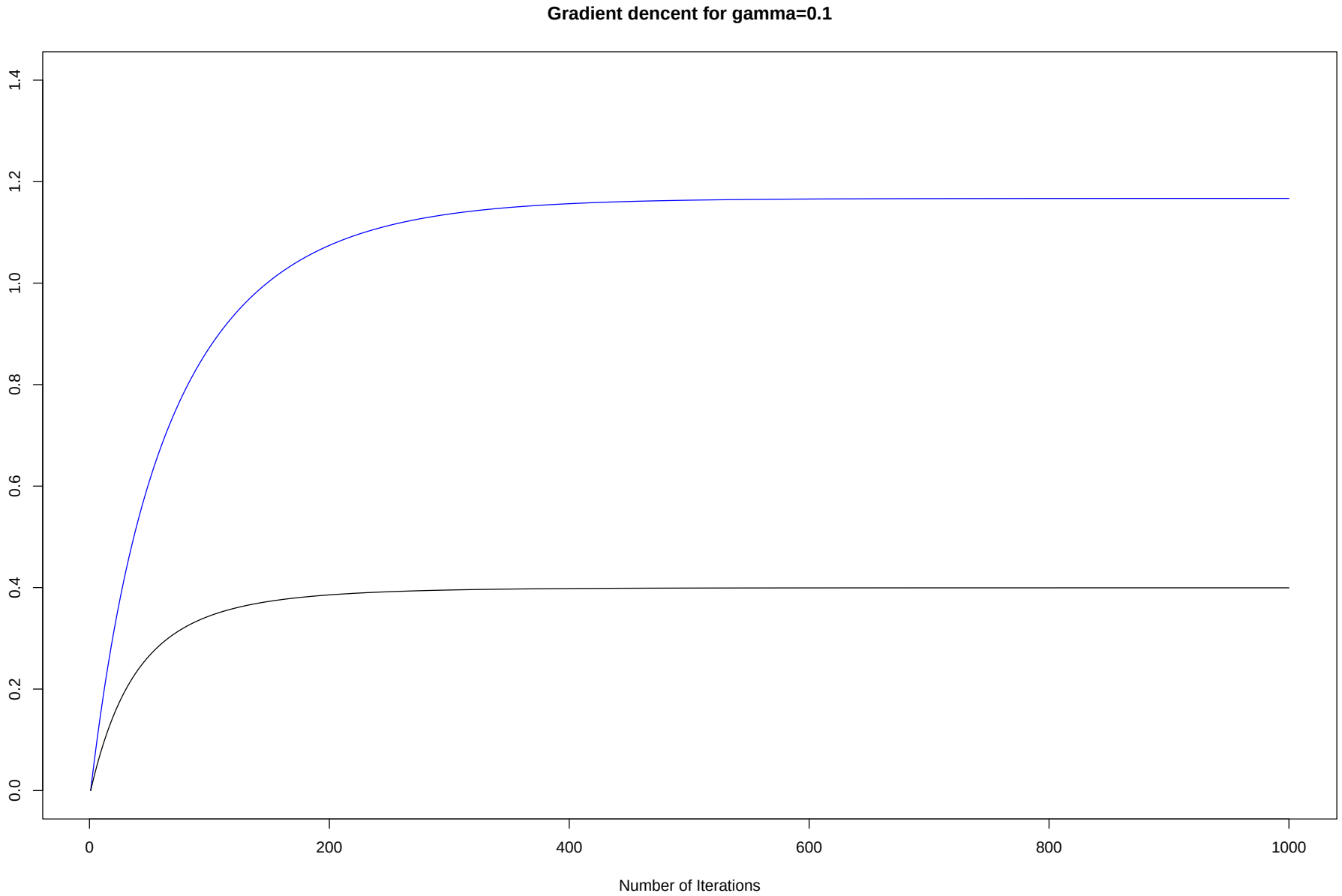
Gradient Dencent算法

```
###设定初始值£(0,0)£以及步长\gamma=0.01, 我们可以得到逻辑回归的数值解
X1<-matrix(1, nrow =n,ncol =2)
X1[,2]<-X
## gradient dencent
gam=0.01
m=10000
AA<-matrix(0,nrow =2,ncol=m)
for (k in 1:(m-1))
{beta0<-AA[,k]
pr<-as.vector(exp(X1%*%beta0)/(1+exp(X1%*%beta0))-Y)
beta1<-beta0-gam*apply(diag(pr)%*%X1, 2,mean)
AA[,k+1]=beta1
}
```

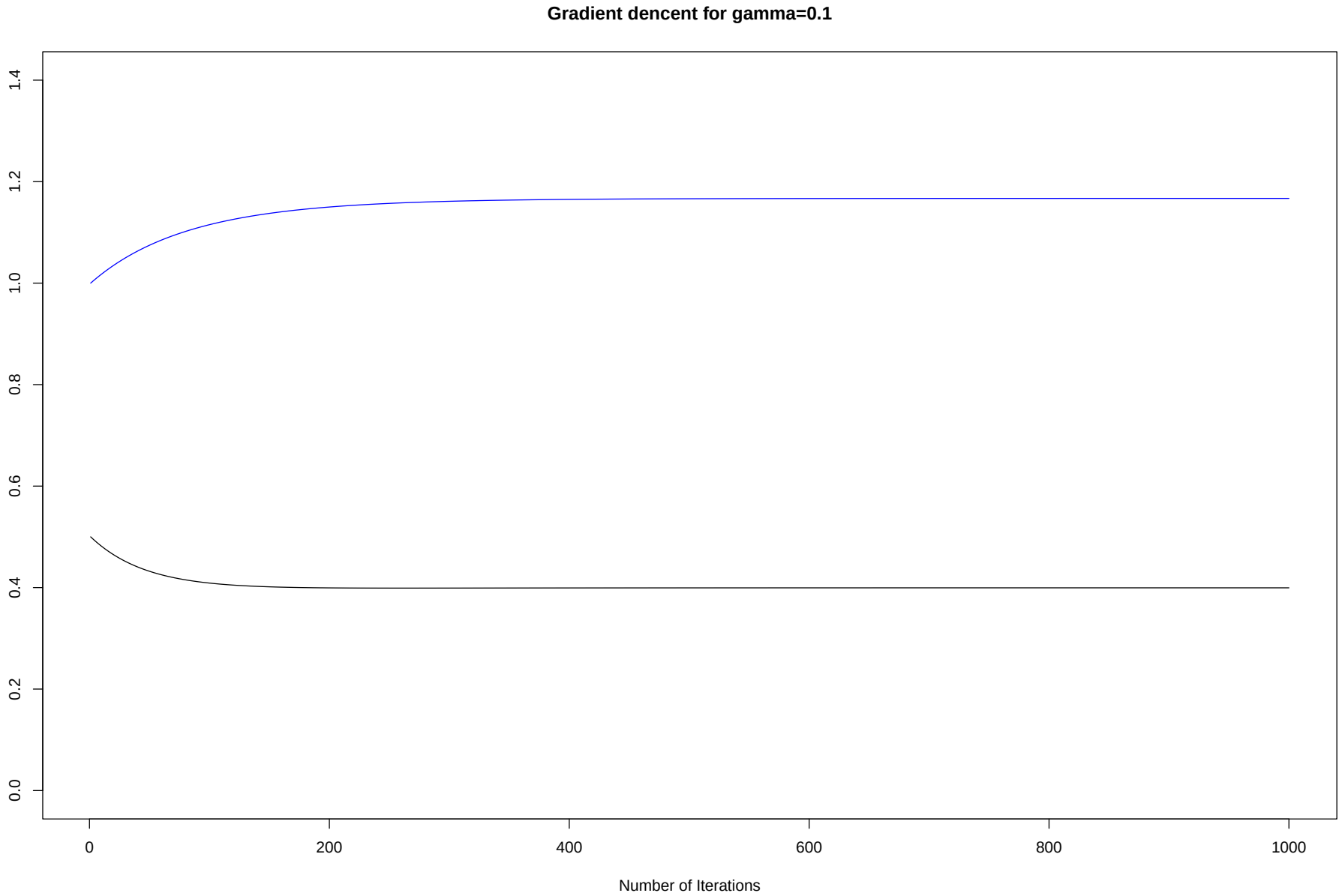
Gradient Dencent迭代结果



Gradient Dencent-步长0.1



Gradient Dencent-初始值$(0, 5, 1)$



Gradient Dencent算法

```
##在算法上设置好的初始值以及好的步长，可以让算法快速收敛到稳定的状态中，实际操作中，一般通过判断两次迭代之间的差异非常小来停止迭代，例如£\|x_{n+1}-x_{n}\|\leq 1e-6£.
gdlogit<-function(X1,Y,beta0=NULL,gam=0.1,epsilon=1e-6)
{
n=nrow(X1);
p=ncol(X1);
  if (is.null(beta0)){beta0=rep(0,p)}
beta=beta0
beta1=beta0
ep=1;
while (ep>epsilon) {
  beta=beta1
  pr<-as.vector(exp(X1%*%beta)/(1+exp(X1%*%beta))-Y)
  beta1<-beta-gam*apply(diag(pr)%*%X1, 2,mean)
  ep<-sqrt(sum((beta1-beta)^2))
}
return(beta1)
}
gdlogit(X1,Y)
```

```
## [1] 0.3995066 1.1668613
```

```
##基于R自带的glm函数可以得到逻辑回归的估计为：
lr_beta=coef(glm(Y~X,family ='binomial'))
t(lr_beta)
```

```
##      (Intercept)        X
## [1,]   0.3995191 1.166952
```

逻辑回归算法-Newton's method

Newton's method常见形式是求解方程$g(x)=0$，对于优化问题

$$\arg\min f(x),$$

Newton's method考虑的问题为

$$\nabla f(x)=0$$

迭代过程为:

$$x_{n+1}=x_n-(\nabla^2 f(x_n))^{-1}\nabla f(x_n),$$

其中$\nabla^2 f(x)$为函数的Hessian矩阵, 记$x=(X_1,\ldots,X_p)'$

$$\nabla^2 f(x)=(\frac{\partial^2 f(x)}{\partial X_i \partial X_j})_{p\times p}.$$

逻辑回归算法-Newton's method

对于逻辑回归问题

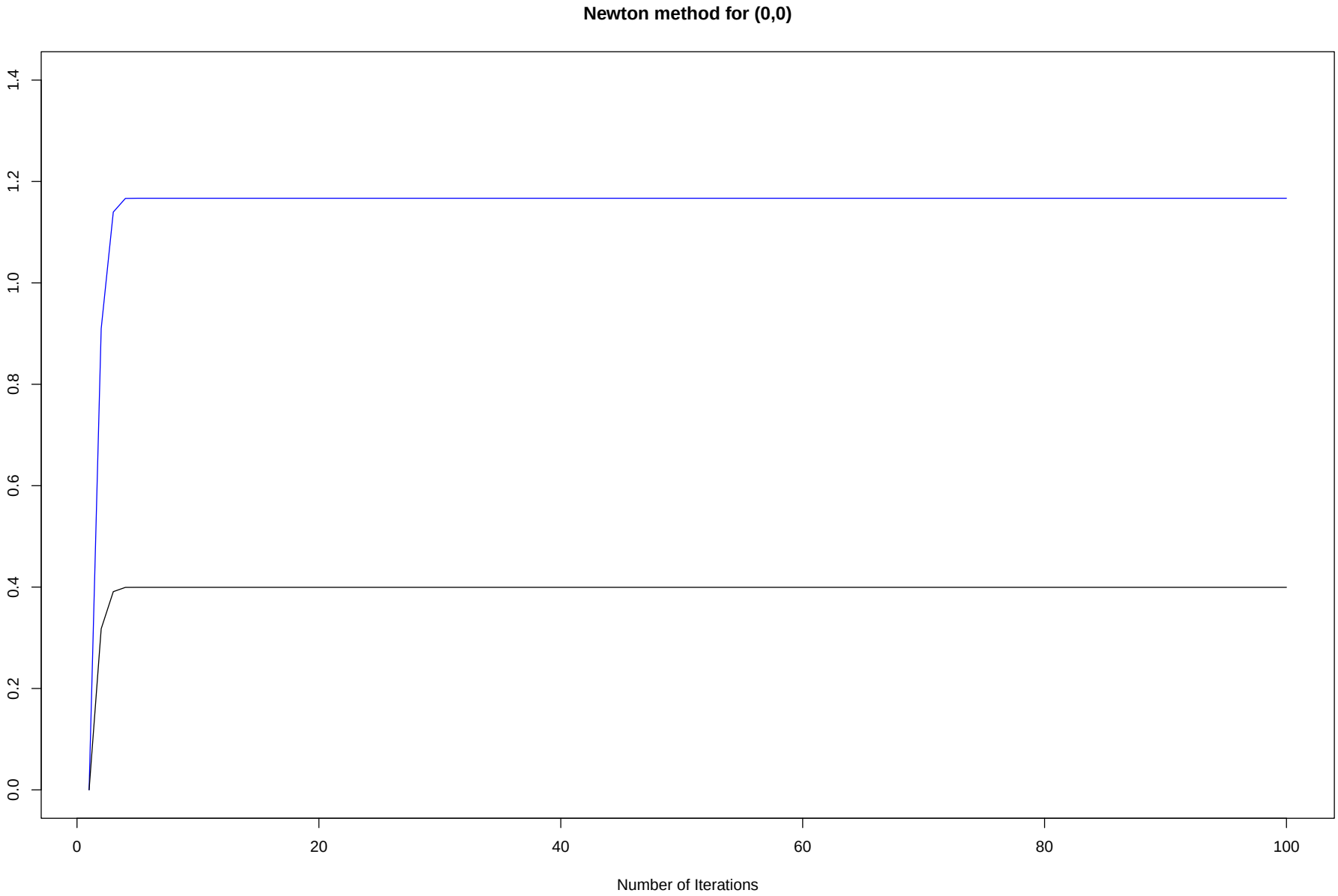
$$\text{Gradient} = \begin{pmatrix} \frac{1}{n}\sum_{i=1}^{n}\{\frac{e^{\beta_0+X_i'\beta_1}}{1+e^{\beta_0+X_i'\beta_1}} - Y_i\} \\ \frac{1}{n}\sum_{i=1}^{n}\{\frac{e^{\beta_0+X_i'\beta_1}}{1+e^{\beta_0+X_i'\beta_1}} - Y_i\}X_i \end{pmatrix},$$

$$\text{Hessian} = \begin{pmatrix} \frac{1}{n}\sum_{i=1}^{n}\frac{e^{\beta_0+X_i'\beta_1}}{(1+e^{\beta_0+X_i'\beta_1})^2} & \frac{1}{n}\sum_{i=1}^{n}\frac{e^{\beta_0+X_i'\beta_1}}{(1+e^{\beta_0+X_i'\beta_1})^2}X_i \\ \frac{1}{n}\sum_{i=1}^{n}\frac{e^{\beta_0+X_i'\beta_1}}{(1+e^{\beta_0+X_i'\beta_1})^2}X_i & \frac{1}{n}\sum_{i=1}^{n}\frac{e^{\beta_0+X_i'\beta_1}}{(1+e^{\beta_0+X_i'\beta_1})^2}X_i^2 \end{pmatrix} \geq 0.$$

Newton's method

```
m=100
AA<-matrix(0,nrow =2,ncol=m)
for (k in 1:(m-1))
{
beta0<-AA[,k]
pr<-as.vector(exp(X1%*%beta0)/(1+exp(X1%*%beta0)))
gradient<-apply(diag(pr-Y)%*%X1, 2,mean);
Hessian<-t(X1)%*%diag(pr*(1-pr))%*%X1/n
beta1<-beta0-solve(Hessian,gradient)
AA[,k+1]=beta1
}
```

Newton's method迭代结果



Newton's method算法

```
nrlogit<-function(X1,Y,beta0=NULL,gam=0.1,epsilon=1e-6)
{
  n=nrow(X1);
  p=ncol(X1);
  if (is.null(beta0)){beta0=rep(0,p)}
  beta=beta0
  beta1=beta0
  ep=1;
  while (ep>epsilon^2) {
    beta=beta1
    pr<-as.vector(exp(X1%*%beta)/(1+exp(X1%*%beta)))
    gradient<-apply(diag(pr-Y)%*%X1, 2,mean);
    Hessian<-t(X1)%*%diag(pr*(1-pr))%*%X1/n
    beta1<-beta-solve(Hessian,gradient)
    ep<-sum((beta1-beta)^2)
  }
  return(beta1)
}
nrlogit(X1,Y)

## [1] 0.3995191 1.1669519

t(lr_beta)

##      (Intercept)        X
## [1,]   0.3995191 1.166952
```

逻辑回归的相合性

```
## 系数估计与样本量关系：在上述例子中，真实的参数是£(0,1)£,而估计给出的结果并不是非常精确，这主要是由于样本量不够大造成的，为了探究这一现象，我们可以看看随着训练数据量的增大，估计的效果：
ff<-function(n){
X<-rnorm(n)   ##生成解释变量
Y=X   ##初始化响应变量
for (k in 1:n)
{
  pr=exp(X[k])/(1+exp(X[k]))
  Y[k]=rbinom(1,1,pr)
}
beta<-as.vector(coef(glm(Y~X,family ='binomial')))
return(beta)
}
ff(100)

## [1] -0.3985318  0.7823254

ff(500)

## [1] -0.07952253  0.88459540

ff(1000)

## [1] 0.01699284 0.98660165

ff(10000)

## [1] 0.02404445 1.04464173
```

Section 5

损失函数

损失函数–loss function

"In mathematical optimization and decision theory, a loss function or cost function (sometimes also called an error function)is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function. An objective function is either a loss function or its opposite (in specific domains, variously called a reward function, a profit function, a utility function, a fitness function, etc.), in which case it is to be maximized.

In statistics, typically a loss function is used for parameter estimation, and the event in question is some function of the difference between estimated and true values for an instance of data. The concept, as old as Laplace, was reintroduced in statistics by Abraham Wald in the middle of the 20th century. In the context of economics, for example, this is usually economic cost or regret. In classification, it is the penalty for an incorrect classification of an example. In actuarial science, it is used in an insurance context to model benefits paid over premiums, particularly since the works of Harald Cramér in the 1920s. In optimal control, the loss is the penalty for failing to achieve a desired value. In financial risk management, the function is mapped to a monetary loss."

—Wikipedia

损失函数–toy example

单位面积圆半径

已知$\pi \approx 3.1415926$, 构造一个面积为1的单位圆，即计算半径

$$r = \frac{1}{\sqrt{\pi}}.$$

假定只能用加减乘除运算，如何计算r? 或者判断下列哪个半径更好:

$$A.\ r = 0.5, \quad B.\ r = 0.55, \quad C.\ r = 0.6.$$

损失函数

对任意的$r > 0$,定义损失函数(构造原理?)

$$L_1(r) = (\pi r^2 - 1)^2;$$
$$L_2(r) = \frac{1}{3}\pi r^3 - r = \frac{1}{3} r(\pi r^2 - 3).$$

损失函数

对任意的$r > 0$,定义损失函数(构造原理?)

$$L_1(r) = (\pi r^2 - 1)^2;$$
$$L_2(r) = \frac{1}{3}\pi r^3 - r = \frac{1}{3} r(\pi r^2 - 3).$$

应用

- 对于任意的$r_1, r_2 > 0$, 我们可以通过计算$L(r_1) >< L(r_2)$来比较好坏.
- 从优化的角度，可以通过求解优化的方式得到最优解

$$\arg\min_{r>0} L_1(r) = \arg\min_{r>0} L_2(r) = \frac{1}{\sqrt{\pi}}.$$

Newton's method

对于优化问题

$$\arg\min f(x)$$

或者等价的方程问题

$$f'(x) = 0.$$

Newton's method

对于优化问题

$$\arg\min f(x)$$

或者等价的方程问题

$$f'(x) = 0.$$

Isaac Newton在1669年给出了迭代过程:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

即给定一个初始值x_0，我们可以迭代得到$x_1, x_2, \cdots$.

Newton's method

对于优化问题

$$\arg\min f(x)$$

或者等价的方程问题

$$f'(x) = 0.$$

Isaac Newton在1669年给出了迭代过程:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

即给定一个初始值x_0， 我们可以迭代得到$x_1, x_2, \cdots$.

数学原理 : $f'(x) \approx f'(x_n) + f''(x_n)(x - x_n) = 0 \implies x = x_n - \frac{f'(x_n)}{f''(x_n)}.$

Newton's method计算$1/\sqrt{\pi}$

损失函数$L_1(r) = (\pi r^2 - 1)^2$:

$$r_{n+1} = r_n - \frac{\pi r_n^3 - r_n}{3\pi r_n^2 - 1} = \frac{2\pi r_n^3}{3\pi r_n^2 - 1}.$$

损失函数$L_2(r) = \frac{1}{3}\pi r^3 - r$:

$$r_{n+1} = r_n - \frac{\pi r_n^2 - 1}{2\pi r_n} = \frac{1}{2} r_n + \frac{1}{2\pi r_n}$$

Newton's method计算$1/\sqrt{\pi}$

```
f1<-function(x){return(2*pi*x^3/(3*pi*x^2-1))}
f2<-function(x) {return(x/2+1/(2*pi*x))}
m=10
r1<-rep(1,m)
r2<-rep(0.1,m)
r3<-rep(1,m)
r4<-rep(0.1,m)
for (k in 2:m)
{r1[k]=f1(r1[k-1]);
r2[k]=f1(r2[k-1]);
r3[k]=f2(r3[k-1]);
r4[k]=f2(r4[k-1]);}
r1
```

```
##  [1] 1.0000000 0.7457983 0.6144018 0.5697421 0.5642697 0.5641896 0.5641896
##  [8] 0.5641896 0.5641896 0.5641896
```

```
r2
```

```
##  [1]   1.000000e-01  -6.936980e-03   2.098400e-06  -5.805567e-17   1.229458e-48
##  [6] -1.167674e-143   0.000000e+00   0.000000e+00   0.000000e+00   0.000000e+00
```

```
r3
```

```
##  [1] 1.0000000 0.6591549 0.5710305 0.5642306 0.5641896 0.5641896 0.5641896
##  [8] 0.5641896 0.5641896 0.5641896
```

```
r4
```

```
##  [1] 0.1000000 1.6415494 0.9177288 0.6322870 0.5678566 0.5642014 0.5641896
##  [8] 0.5641896 0.5641896 0.5641896
```

线性回归–Summary

- Step 1: 对于任意一条线(任意一个β), 可以得到预测值:

$$\hat{y}_i = \mathbf{X}_i^{\mathsf{T}}\beta;$$

- Step 2: 对于预测值$\hat{y}_i$和实际观察值y_i,定义平方损失函数

$$L(y, \hat{y}) = (y - \hat{y})^2.$$

- Step 3: 最小二乘法

$$\arg\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{y}_i)^2 = (\mathbb{X}^{\mathsf{T}}\mathbb{X})^{-1}\mathbb{X}^{\mathsf{T}}\mathbb{Y}.$$

逻辑回归–Summary

给定样本$(\mathbf{X}_1, y_1), \cdots, (\mathbf{X}_n, y_n)$:

- Step 1: 对于任意一条线(任意一个β), 可以得到预测值:

$$\hat{y}_i = \frac{e^{\mathbf{X}_i^\top \beta}}{1 + e^{\mathbf{X}_i^\top \beta}} \in (0, 1).$$

- Step 2: 定义交叉熵损失

$$L(y, \hat{y}) = -\{y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\}.$$

- Step 3: 逻辑回归模型

$$\arg\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{y}_i)^2.$$

Thank you !