Machine Learning 2019

袁欣

2019 年 3 月 10 日

1 线性模型

1.1 西瓜数据

从 csv 文件中读取西瓜数据，并进行展示。

- 代码如下：

```
wmda <- read.csv(file = "data/西瓜数据3.0a.csv")
wmda$label <- as.factor(wmda$label)
```

- 数据展示：

```
knitr::kable(head(wmda))
```

idx	density	sugar	label
1	0.697	0.460	1
2	0.774	0.376	1
3	0.634	0.264	1
4	0.608	0.318	1
5	0.556	0.215	1
6	0.403	0.237	1

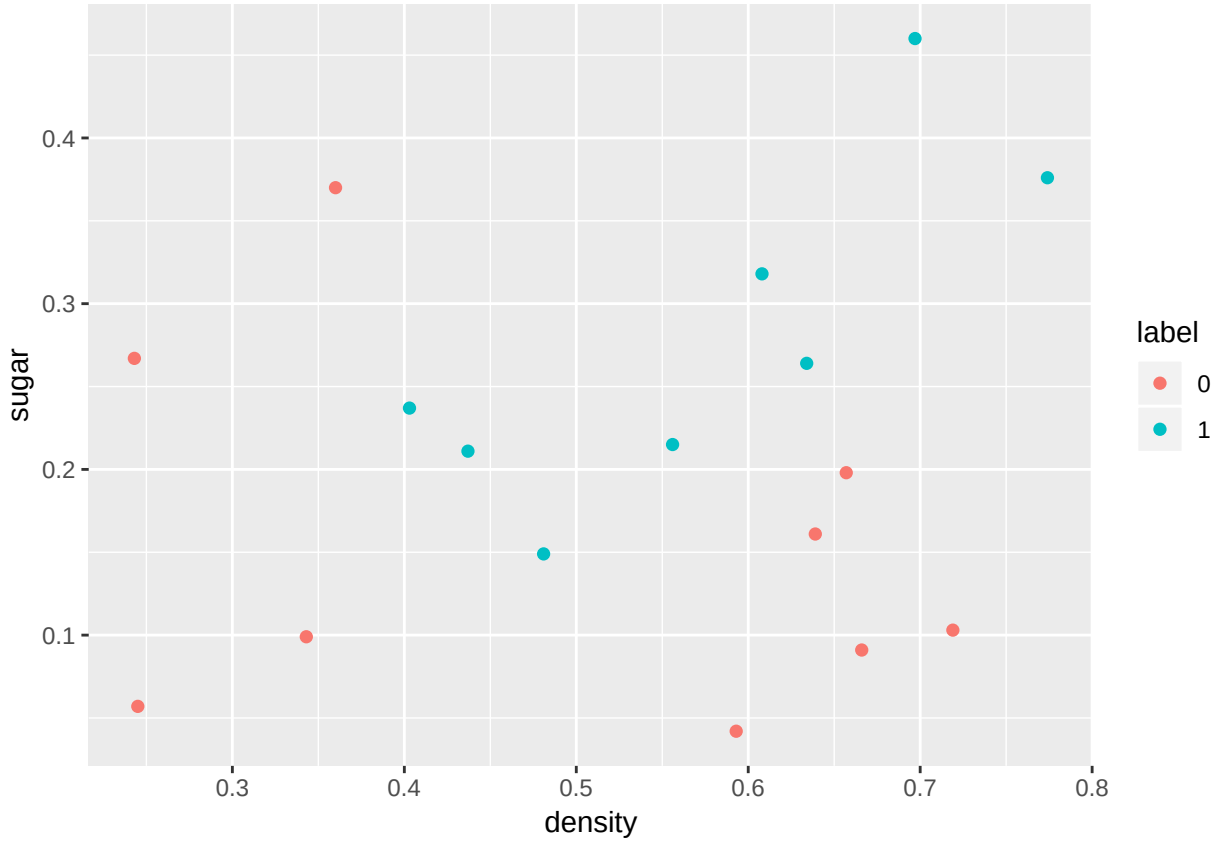
```
summary(wmda[, -1])
```

```
##     density           sugar        label
##  Min.   :0.2430   Min.   :0.0420   0:9
##  1st Qu.:0.4030   1st Qu.:0.1030   1:8
##  Median :0.5930   Median :0.2110
##  Mean   :0.5326   Mean   :0.2128
##  3rd Qu.:0.6570   3rd Qu.:0.2670
```

```
##  Max.   :0.7740   Max.   :0.4600
```

- 画图：

```
ggplot(data = wmda, aes(x = density, y = sugar, color = label)) +
  geom_point(size = 2.0, shape = 16)
```



1.2 逻辑回归简介

1.2.1 分类问题

逻辑回归是处理典型分类问题的，如判断一个邮件是否为垃圾邮件 (Spam / not Spam)。逻辑回归与线性回归直观区别是：

线性回归将预测值映射到了实数集 ($h_\theta(x)$ can be > 1 or < 0); 逻辑回归的预测值仅在 $[0, 1]$ 之间 ($0 \leq h_\theta(x) \leq 1$)。

1.2.2 假设表示

(Hypothesis Representation) 假设表示代表当有一个实际问题时，我们用什么样的方程表示。逻辑回归是想实现 $0 \leq h_\theta(x) \leq 1$。

那么假设函数就可以是如下形式：

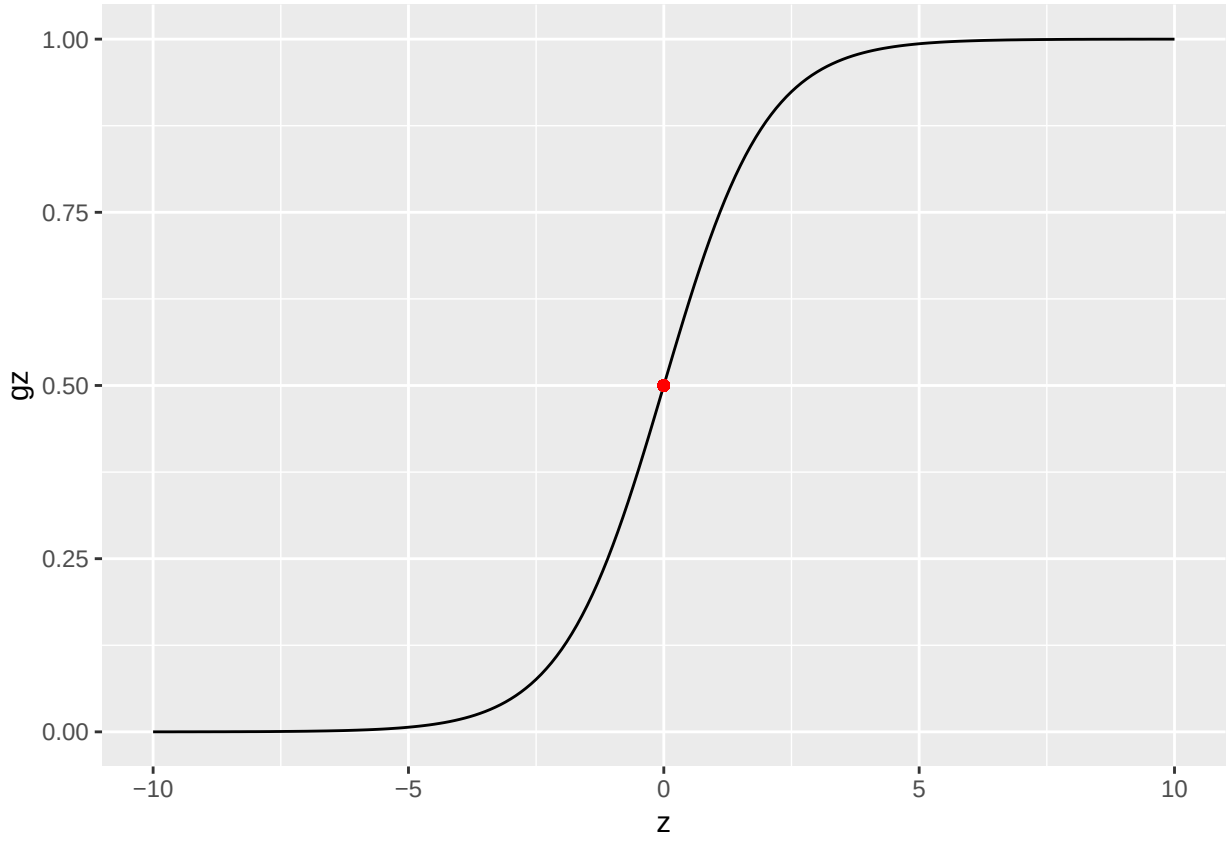
$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$h_\theta(x) = P(y=1|x;\theta)$$

$h_\theta(x)$ 的含义为 (Probability that $y=1$,given x, parameterized by θ)

- $g(z)$ 的图像如下：



1.2.3 决策边界

假设当 $h_\theta(x)$ 的值大于等于 0.5，预测值 $y=1$; 当 $h_\theta(x)$ 的值小于 0.5 时，预测值 $y=0$。决策边界就是 $h_\theta(x)=0.5$，也就是 $\theta^T x = 0$。例如当求得参数 $\theta=[-3;1;1]$，则决策边界为 $x_1+x_2=3$。

非线性决策边界是指当假设函数 $h_\theta(x)$ 为非线性函数时的决策边界。例如当 $h_\theta(x)=g(\theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_1^2+\theta_4x_2^2)$，$\theta=[-1;0;0;1;1]$，此时的决策边界为 $x_1^2+x_2^2=0$。

1.2.4 代价函数

假设测试集为 $\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),...,(x^{(m)},y^{(m)})\}$，共 m 个样本。$x \in [x_0;x_1;...;x_n]$，其中 $x_0=1$，$y \in \{0,1\}$。线性的回归的代价函数为：

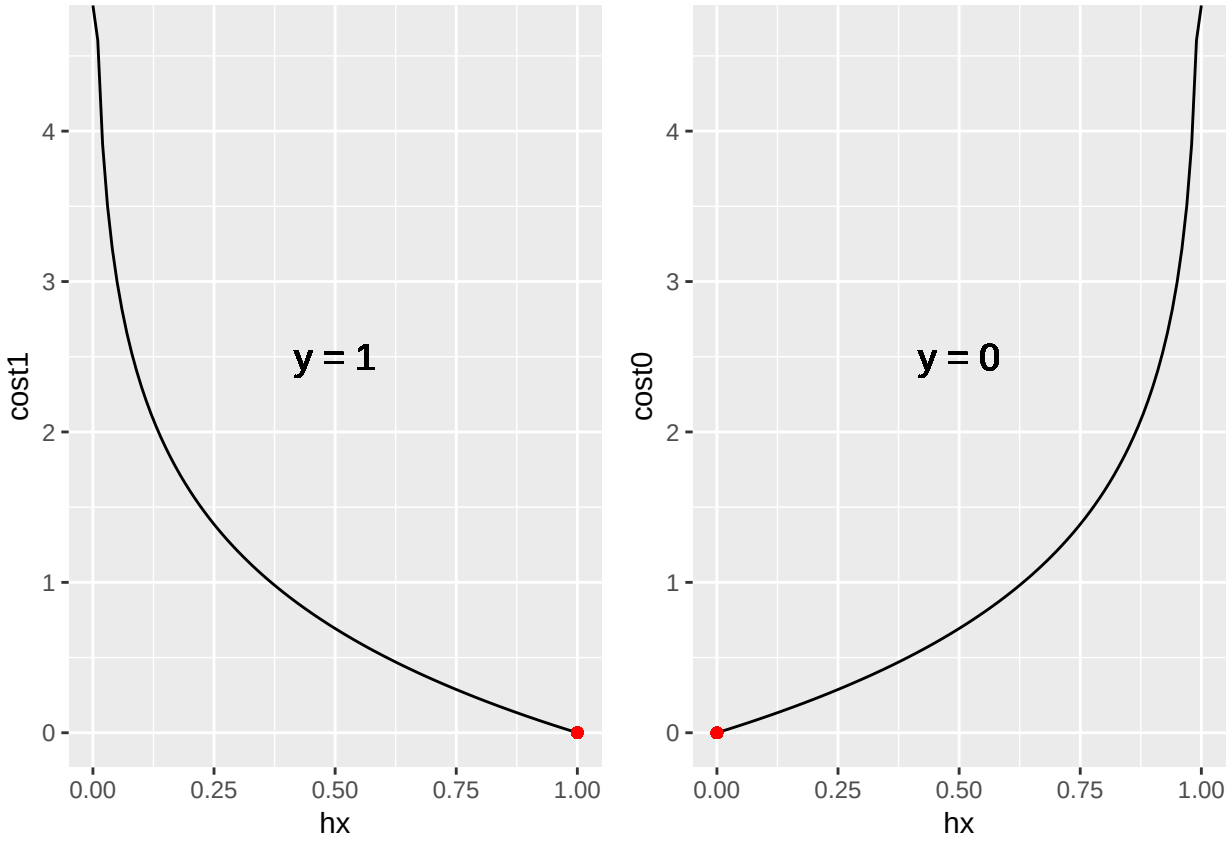
$$J_\theta = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(h_\theta(x^{(i)})-y^{(i)})^2$$

$$cost(h_\theta(x),y) = \frac{1}{2}(h_\theta(x)-y)^2$$

然而这个函数在逻辑回归里是 θ 的非凸函数。逻辑回归的代价函数如下式时为凸函数

$$cost(h_\theta(x),y) = \begin{cases} -log(h_\theta(x)), & \text{if } y{=}1 \\ -log(1-h_\theta(x)), & \text{if } y{=}0 \end{cases}$$

- 代价函数的图形如下：



简化形式为：

$$cost(h_\theta(x),y) = -ylog(h_\theta(x)) - (1-y)log(1-h_\theta(x))$$

那么逻辑回归的代价函数就为：

$$J_\theta = \frac{1}{m}\sum_{i=1}^{m}(-y^{(i)}log(h_\theta(x^{(i)})) - (1-y^{(i)})log(1-h_\theta(x^{(i)})))$$

上式来自于极大似然估计，具体推到过程如下：从假设表示中有提到 $h_\theta(x)$ 的含义为 (Probability that $y = 1$,given x, parameterized by θ)。利用极大似然法（即令每个样本属于其真实标记的概率越大越好）估计 θ。似然函数为：

$$L = \prod_{i=1}^{m} h_\theta(x^{(i)})^{Y^{(i)}}(1 - h_\theta(x^{(i)}))^{1-Y^{(i)}}$$

$$lnL = \sum_{i=1}^{m} Y^{(i)}(ln(h_\theta(x^{(i)}))) + (1 - Y^{(i)})(ln(1 - h_\theta(x^{(i)})))$$

求似然函数的最大值等价于求 J_θ 的最小值。

1.2.5 梯度下降法

上节中已知 J_θ，对 θ 求偏导数得：

$$\begin{aligned} J_\theta &= \frac{1}{m}\sum_{i=1}^{m}(-y^{(i)}log(h_\theta(x^{(i)})) - (1 - y^{(i)})log(1 - h_\theta(x^{(i)}))) \\ &= \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}(log(h_\theta(x^{(i)})) - log(1 - h_\theta(x^{(i)}))) - log(1 - h_\theta(x^{(i)})) \\ &= \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}log\frac{h_\theta(x^{(i)})}{1 - h_\theta(x^{(i)})} - log(1 - h_\theta(x^{(i)})) \\ &= \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}(\theta^T x^{(i)}) + log(1 + e^{\theta^T x^{(i)}}) \end{aligned}$$

$$\frac{\partial}{\partial\theta_j}J_\theta = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

- 编程实现逻辑回归的梯度下降算法

```
sigmoid <- function(z){
  return(1 / (1 + exp(-z)))
}
gradientDescent <- function(X, y, theta, alpha, num_iters){
  # Initialize some useful values
  m <- length(y) # number of training examples
  J.history <- rep(0, num_iters)
  for(i in 1:num_iters){
    theta <- theta - (1/m) * (t(X) %*% (sigmoid(X %*% theta) - y))
    #  Save the cost J in every iteration
    J.history[i] <- (1/m) *
      sum(-y * log(sigmoid(X %*% theta)) -
            (1-y) * (log(1 - sigmoid(X %*% theta))))
  }
```

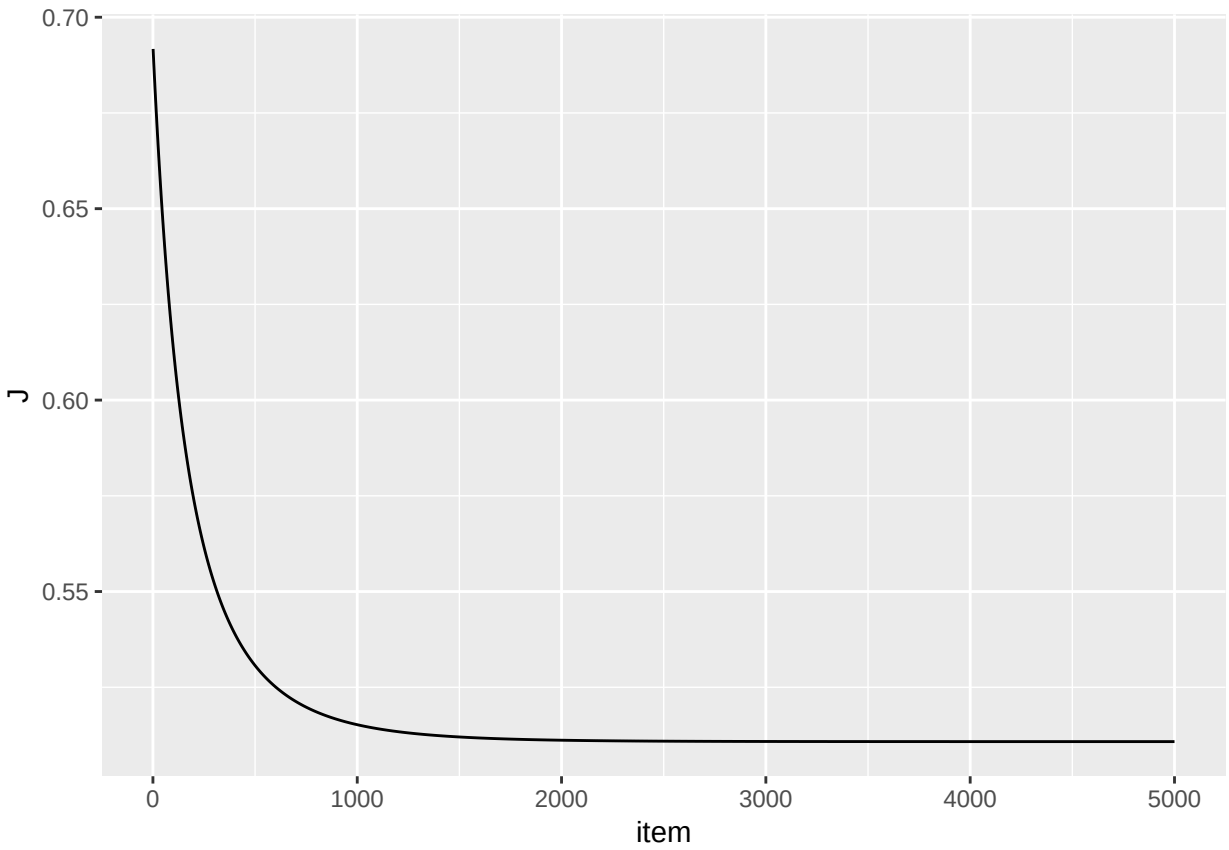
```
  return(list(theta = theta, J = J.history))
}

X <- as.matrix(wmda[, 2:3])
X <- cbind(1, X)
y <- as.matrix(as.numeric(as.character(wmda$label)))
initial_theta <- matrix(rep(0, 3), ncol = 1)
alpha <- 0.1
Ret <- gradientDescent(X, y, initial_theta, alpha,
                       num_iters = 5000)
```

- 回归参数与代价函数曲线如下：

```
as.numeric(Ret$theta)
```

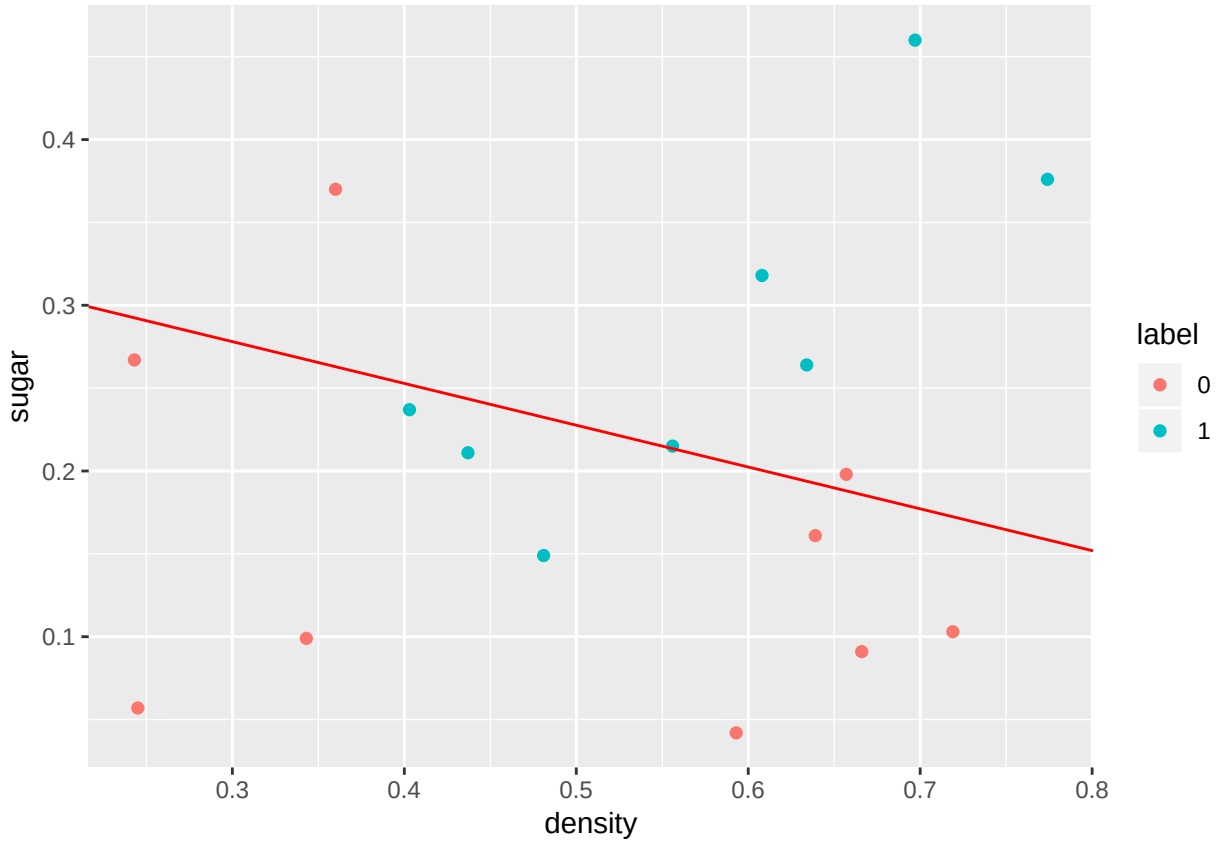
```
## [1] -4.419850  3.151482 12.495210
```

```
ggplot(data = data.frame(item = 1:length(Ret$J), J = Ret$J),
       aes(x = item, y = J)) + geom_line()
```



- 决策边界如下：

```r
ggplot(data = wmda, aes(x = density, y = sugar, color = label)) +
  geom_point(size = 2.0, shape = 16) +
  geom_abline(slope = -Ret$theta[2] / Ret$theta[3],
              intercept = -Ret$theta[1]/Ret$theta[3],
              color = "red")
```



1.2.6 mapFeature

从上图可以看到，逻辑回归对西瓜集的分类是较差的。我们也可以直观的看到西瓜集是线性不可分的！所以这里引入了高阶特征，构建非线性边界去划分西瓜集。

构建方法为选择最高次数，将两变量映射到高阶上形成新特征。例如构建最高幂次为 6 的特征，此时会产生新特征如：x_1^6、x_2^6、$x_1^5x_2$、……、x_1x_2、x_2、x_1 共 28 个特征。

- 构建函数

```r
mapFeature <- function(x1, x2, degree){
  df <- matrix(1, nrow = length(x1))
  for(i in 1:degree){
    for(j in 0:i){
      x <- x1^(i - j) * x2^(j)
```

```
      df <- cbind(df, x)
    }

  }
  return(df)
}
x1 <- wmda$density
x2 <- wmda$sugar
X <- mapFeature(x1, x2, 6)
```

1.2.7 正则化

如果我们有太多特征，那么通过训练集得到的模型很可能会过拟合，使得模型对新样本的预测值差。为了解决过拟合的问题，提出了正则化（Regularization）。过拟合的问题还可以通过手工选择特征或者通过算法（如 PCA）来减少特征数。

正则化的思想是控制参数 θ_j 的大小来防止过拟合。一般来说不对 θ_0 进行正则化。

正则化后的代价函数与偏导数如下所示：

$$J_\theta = \frac{1}{m}\sum_{i=1}^{m}(-y^{(i)}log(h_\theta(x^{(i)})) - (1-y^{(i)})log(1-h_\theta(x^{(i)}))) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$\frac{\partial}{\partial\theta_j}J_\theta = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

- 编程实现正则化逻辑回归的梯度下降算法

```
gradientDescent <- function(X, y, theta, alpha, num_iters, lambda){
  # Initialize some useful values
  m <- length(y) # number of training examples
  n <- ncol(X)
  J.history <- rep(0, num_iters)
  for(i in 1:num_iters){
    theta[1] <- theta[1] -
      (1/m) * (t(X[, 1]) %*% (sigmoid(X %*% theta) - y))
    theta[2:n] <- theta[2:n] -
      (1/m) * (t(X[, 2:n]) %*% (sigmoid(X %*% theta) - y)) +
      lambda/m * theta[2:n]
    #  Save the cost J in every iteration
    J.history[i] <- (1/m) *
      sum(-y * log(sigmoid(X %*% theta)) -
            (1-y) * (log(1 - sigmoid(X %*% theta)))) +
```

```
      (lambda/2/m) * sum(theta[2:n] ^2)
  }
  return(list(theta = theta, J = J.history))
}

y <- as.matrix(as.numeric(as.character(wmda$label)))
initial_theta <- matrix(rep(0, 28), ncol = 1)
alpha <- 0.1
lambda <- 0
Ret <- gradientDescent(X, y, initial_theta, alpha,
                       num_iters = 100000, lambda)
```

- 回归参数、预测精度、代价函数曲线如下：

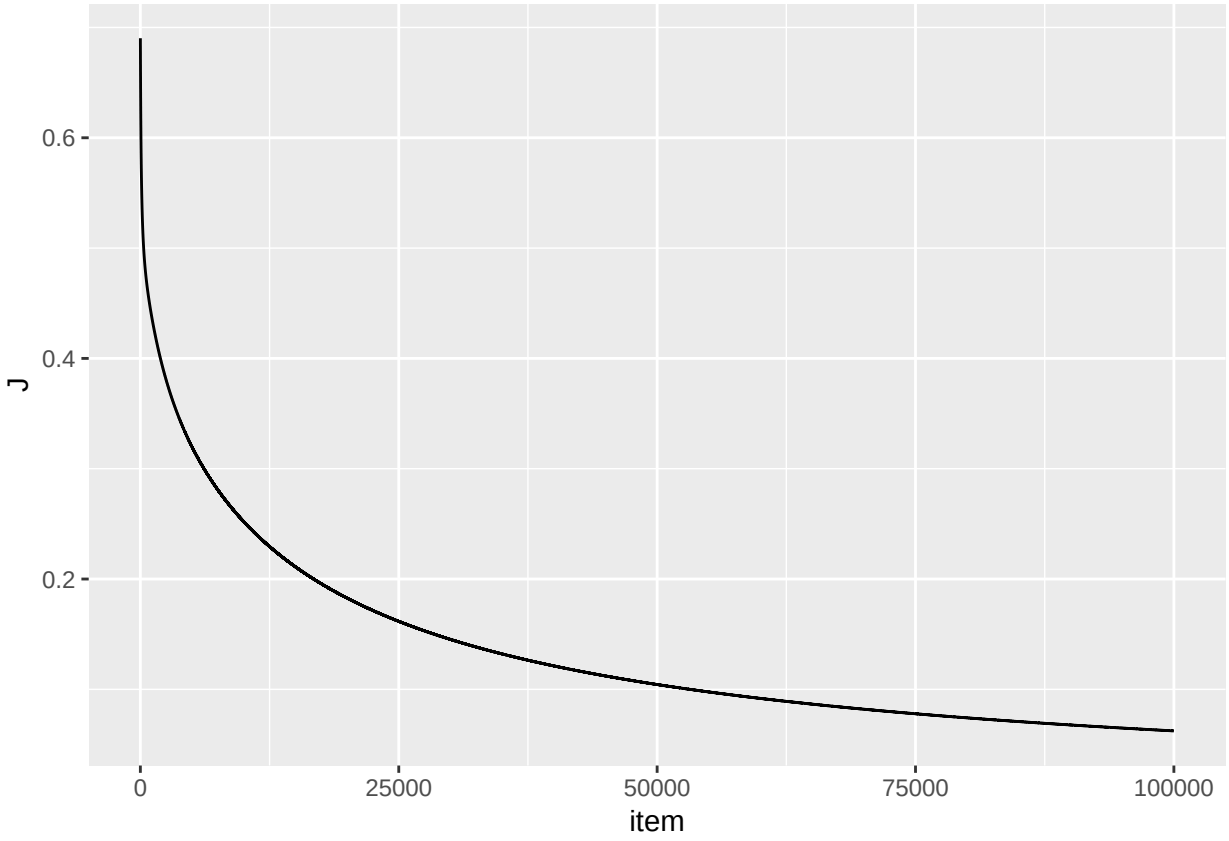
```
as.numeric(Ret$theta)
```

```
##  [1] -18.2576688  55.5141514 -10.7170173 -12.4644799  66.1030944
##  [6] -49.1665058 -48.6936829  60.3009426  14.7737220 -32.2204811
## [11] -50.7437367  44.6194226  26.2238727   1.2997743 -15.4884559
## [16] -37.5326066  32.8471126  24.1358494   9.3827915  -0.7082506
## [21]  -6.5816698 -21.6262370  25.2981189  19.6096944   9.8597332
## [26]   3.2328093  -0.5628648  -2.6294491
```

```
p <- sigmoid(X %*% Ret$theta)
pos <- which(p >= 0.5)
neg <- which(p < 0.5)
p[pos] <- 1
p[neg] <- 0
t <- table(p, wmda$label)
print(paste("prediction accuracy = ", sum(t) / sum(diag(t)) * 100,
            "%"))
```

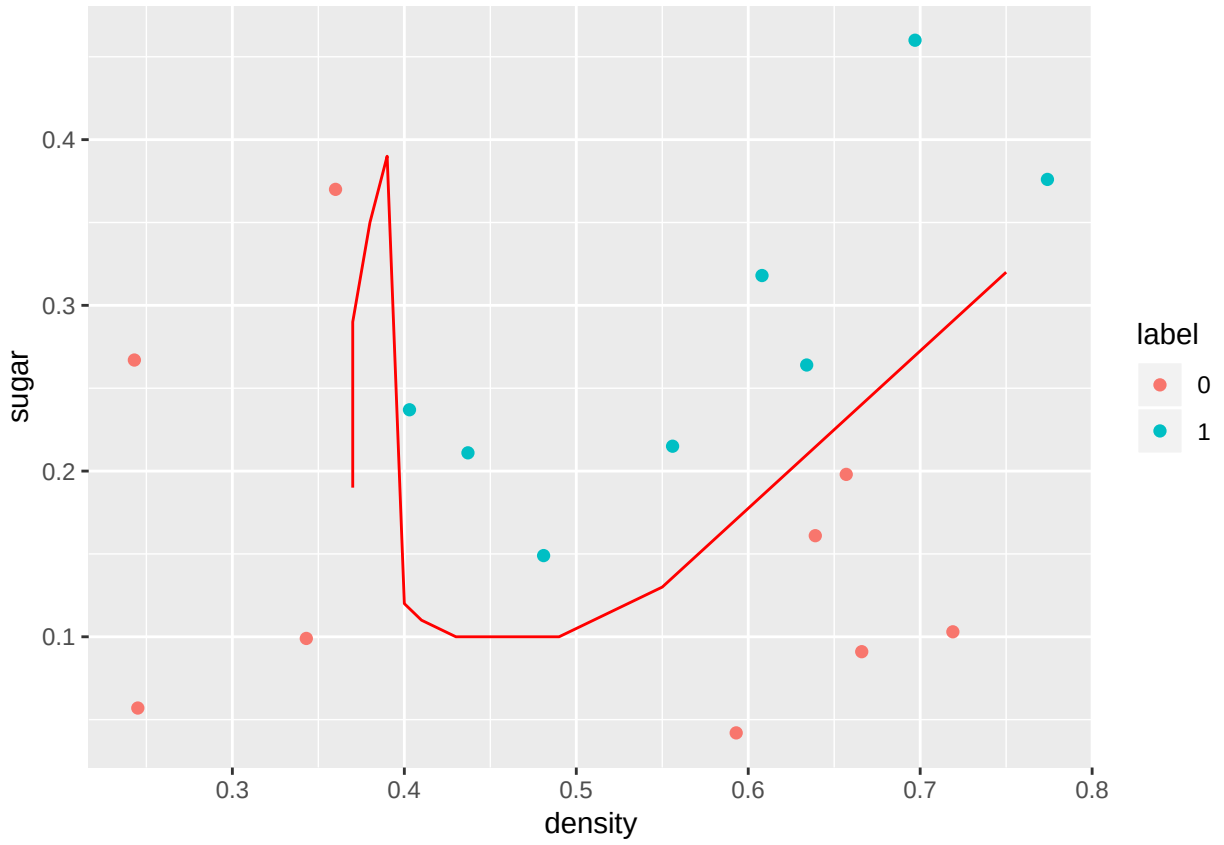
```
## [1] "prediction accuracy =  100 %"
```

```
ggplot(data = data.frame(item = 1:length(Ret$J), J = Ret$J),
       aes(x = item, y = J)) + geom_line()
```



- 非线性决策边界如下：

```
x1 <- seq(0, 0.8, 0.01)
x2 <- seq(0, 0.5, 0.01)
x.grad <- data.frame()
for(i in x1){
  for(j in x2){
    x.grad <- rbind(x.grad, c(i, j))
  }
}
colnames(x.grad) <- c("x1", "x2")
X.grad <- mapFeature(x.grad[, 1], x.grad[, 2], 6)
p <- sigmoid(X.grad %*% Ret$theta)
idx <- which(p < 0.5+0.01 & p > 0.5-0.01)

ggplot(data = wmda, aes(x = density, y = sugar, color = label)) +
  geom_point(size = 2.0, shape = 16) +
  geom_line(data = x.grad[idx, ], aes(x = x1, y = x2),  colour = "red")
```



1.2.8 多分类问题

可以利用 One-vs-all 算法，创建伪训练集，例如：预测天气（Sunny、Cloudy、Rain、Snow），可以学习四个逻辑回归，判断哪个概率最高，则属于哪一类。

1.2.9 利用牛顿法求解

牛顿法主要可以求解方程 $f(\theta)=0$，核心思想是根据泰勒展开式进行迭代求解的。假设 $f(x)=0$ 有近似根 x_k，那么 $f(x)$ 在 x_k 处的泰勒展开式为：

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k)$$

令 $f(x)=0$ 有

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

在求解代价函数最小化的过程中，我们可以利用牛顿法迭代求解一阶偏导数的解，从而得出参数的估计值。也就是：

$$\theta = \theta - \frac{J'(\theta)}{J''(\theta)}$$

一阶偏导数为：

$$\frac{\partial}{\partial \theta_j} J_\theta = \frac{1}{m} \sum_{t=1}^{m} (h_\theta(x^{(t)}) - y^{(t)}) x_j^{(t)}$$

二阶偏导数（Hessian Matrix）为 $n*n$ 的方阵。下面推倒二阶偏导数：

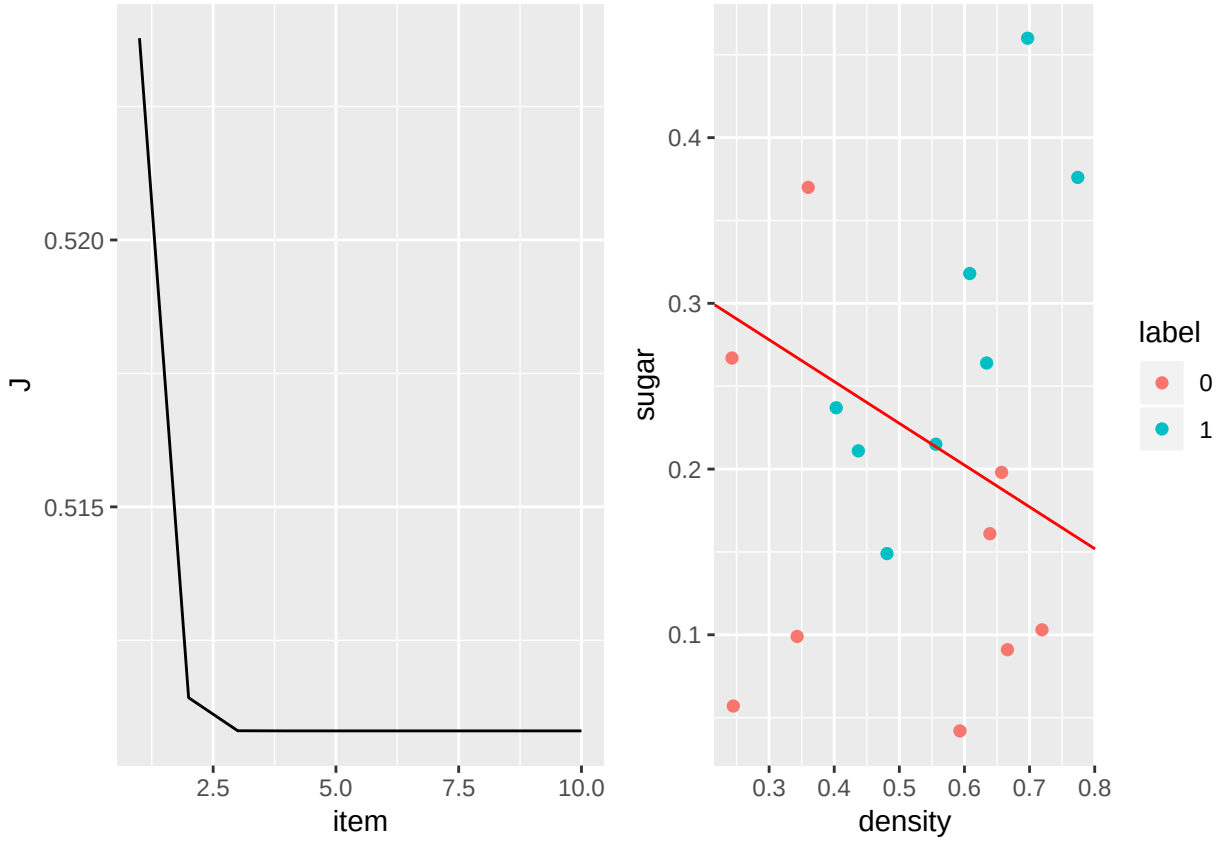
$$\begin{aligned}\frac{\partial^2}{\partial\theta_j\partial\theta_i}J_\theta &= \frac{1}{m}\frac{\partial}{\partial\theta_i}\sum_{t=1}^{m}(h_\theta(x^{(t)})-y^{(t)})x_j^{(t)}\\ &= \frac{1}{m}\sum_{t=1}^{m}x_j^{(t)}\frac{\partial}{\partial\theta_i}h_\theta(x^{(t)})\\ &= \frac{1}{m}\sum_{t=1}^{m}x_j^{(t)}h_\theta(x^{(t)})(1-h_\theta(x^{(t)}))\frac{\partial}{\partial\theta_i}(\theta^Tx^{(t)})\\ &= \frac{1}{m}\sum_{t=1}^{m}x_i^{(t)}x_j^{(t)}h_\theta(x^{(t)})(1-h_\theta(x^{(t)}))\end{aligned}$$

- 代码实现

```
HessianMatrix <- function(X, y, theta, num_iters){
  # Initialize some useful values
  m <- length(y) # number of training examples
  J.history <- rep(0, num_iters)

  for(i in 1:num_iters){
    partial1 <- (1/m) * (t(X) %*% (sigmoid(X %*% theta) - y))
    partial2 <- (1/m) * (t(X) %*% (X * as.numeric(
                           (sigmoid(X %*% theta) *
                           (1 - sigmoid(X %*% theta))))) )
    theta <- theta - solve(partial2) %*% partial1
    #  Save the cost J in every iteration
    J.history[i] <- (1/m) *
      sum(-y * log(sigmoid(X %*% theta)) -
            (1-y) * (log(1 - sigmoid(X %*% theta))))
  }
  return(list(theta = theta, J = J.history))
}
X <- as.matrix(wmda[, 2:3])
X <- cbind(1, X)
initial_theta <- matrix(rep(0, 3), ncol = 1)
Ret <- HessianMatrix(X, y, initial_theta, num_iters = 10)
```

- 回归参数、代价函数曲线、决策边界如下：

```
## [1] -4.428865  3.158330 12.521196
```



1.2.10 牛顿法正则化

- 代码实现

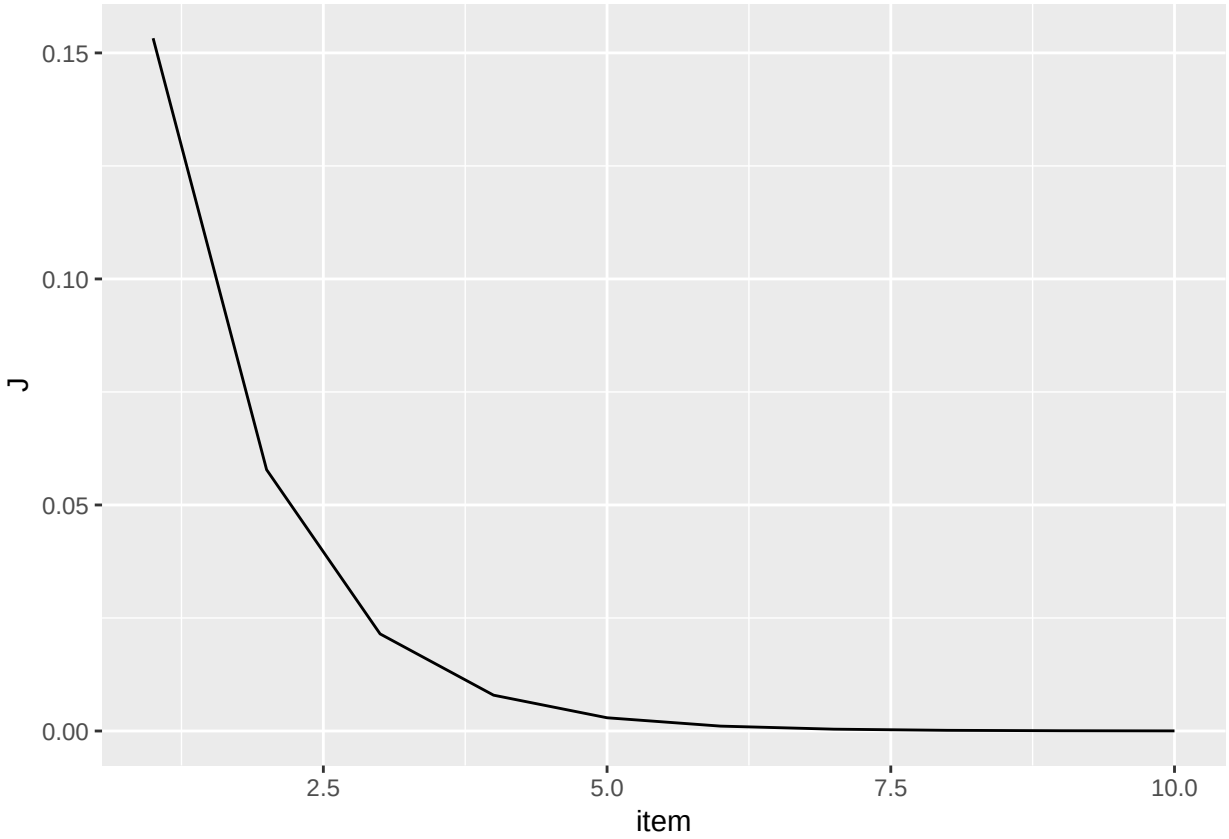
```
HessianMatrix2 <- function(X, y, theta, num_iters, lambda){
  # Initialize some useful values
  m <- length(y) # number of training examples
  n <- ncol(X)
  J.history <- rep(0, num_iters)

  for(i in 1:num_iters){
    partial1 <- matrix(rep(0, n))
    partial1[1] <- (1/m) * (t(X[, 1]) %*% (sigmoid(X %*% theta) - y))
    partial1[2:n] <- (1/m) * (t(X[, 2:n]) %*% (sigmoid(X %*% theta) - y)) +
      lambda/m * theta[2:n]

    partial2 <- (1/m) * (t(X) %*% (X * as.numeric(
                           (sigmoid(X %*% theta) *
                           (1 - sigmoid(X %*% theta))))) )
```

```
    theta <- theta - ginv(partial2) %*% partial1
    #  Save the cost J in every iteration
    J.history[i] <- (1/m) *
      sum(-y * log(sigmoid(X %*% theta)) -
            (1-y) * (log(1 - sigmoid(X %*% theta)))) +
      (lambda/2/m) * sum(theta[2:n] ^2)
  }
  return(list(theta = theta, J = J.history))
}
x1 <- wmda$density
x2 <- wmda$sugar
X <- mapFeature(x1, x2, 6)
y <- as.matrix(as.numeric(as.character(wmda$label)))
initial_theta <- matrix(rep(0, 28), ncol = 1)
lambda <- 0
Ret <- HessianMatrix2(X, y, initial_theta,
                      num_iters = 10, lambda)
```

- 回归参数、代价函数曲线如下：

```
##  [1]   260.58899 -2305.59070   131.21556  5662.94237  -738.76919
##  [6]  -205.44986 -2183.57580  2171.56434  -331.49342   268.51504
## [11] -5259.69998   -94.64807  1782.42515  -318.80433  -889.53148
## [16]  -933.53588 -2313.77204  1750.42685   388.24622 -1015.93544
## [21] -1193.47150  6438.19215 -2860.52426  1047.96320   531.73182
## [26]  -599.01104 -1036.47230  -927.34067
```

```
## [1] "prediction accuracy =  100 %"
```



1.2.11 小结

1. 对比可以发现牛顿法比梯度下降法收敛速度快的多！
2. 在最小化代价函数的过程中还有很多更高级的方法，如 BFGS（共轭梯度）法、L-BGFS 等，它们的优点是不用选择参数 α、收敛速度更快，但是它们也更复杂。
3. 在非线性边界画图中利用的是等值线绘图，也就是将图形分成一个个小的密度点，计算每个密度点的概率值。密度点概率值为 0.5 的等值线即为边界线。但是在实现过程中 `geom_isobands()` 并不能很好实现这个过程。Matlab 可以利用函数 `contour()` 实现，切记在利用这个函数之前将 X 转置。
4. 在 HessianMatrix 矩阵的求逆过程中并没有利用 `solve()` 函数，而是利用了 `MASS` 包里的 `ginv` 函数，当矩阵不可逆时，这个函数求得矩阵伪逆。类似于 Matlab 中 `inv` 与 `pinv` 的关系。

1.3 线性判别分析

线性判别分析（Linear Discriminant Anaysis，简称 LDA）是一种经典的线性学习方法。LDA 的思想非常朴素：给定训练集，设法将样本投射到一条直线上，使得同类样本的投影点尽可能接近、异类样本投影点尽可能远离；在对新样本进行分类时，将其投影到同样的直线上，再根据投影点的位置来确定新样本的类型。

1.3.1 理论基础

将数据点投影到直线 θ^T 上，表达式为

$$y = \theta^T x$$

对于二分类问题，两类样本中心在直线上的投影分别为 $\theta^T\mu_0$ 和 $\theta^T\mu_1$，两类样本投影后的协方差为 $\theta^T\Sigma_0\theta$ 和 $\theta^T\Sigma_1\theta$，具体推到公式为：

$$\begin{aligned}\sum_{x\in D_i}(\theta^T x - \theta^T\mu_i)^2 &= \sum_{x\in D_i}(\theta^T(x-\mu_i))^2 \\ &= \sum_{x\in D_i}\theta^T(x-\mu_i)(x-\mu_i)^T\theta \\ &= \theta^T\sum_{x\in D_i}[(x-\mu_i)(x-\mu_i)^T]\theta \\ &= \theta^T\Sigma_i\theta\end{aligned}$$

欲使同类样本的投影点尽可能接近，可以让同类样本的协方差尽可能的小（类似于方差代表样本离散程度），即 $\theta^T\Sigma_0\theta + \theta^T\Sigma_1\theta$ 尽可能小；而欲使异类样本的投影点尽可能远离，可以让类中心之间的距离尽可能大，即 $||\theta^T\mu_0 - \theta^T\mu_1||_2^2$ 尽可能大。同时考虑这两个方面得到如下最大化优化目标：

$$\begin{aligned}J &= \frac{||\theta^T\mu_0 - \theta^T\mu_1||_2^2}{\theta^T\Sigma_0\theta + \theta^T\Sigma_1\theta} \\ &= \frac{\theta^T(\mu_0-\mu_1)(\mu_0-\mu_1)^T\theta}{\theta^T(\Sigma_0+\Sigma_1)\theta}\end{aligned}$$

定义类内散度矩阵（within-class scatter matrix）：

$$\begin{aligned}S_w &= \Sigma_0 + \Sigma_1 \\ &= \sum_{x\in D_0}(x-\mu_0)(x-\mu_0)^T + \sum_{x\in D_1}(x-\mu_1)(x-\mu_1)^T\end{aligned}$$

定义类间散度矩阵（between-class scatter matrix）：

$$S_b = (\mu_0-\mu_1)(\mu_0-\mu_1)^T$$

则

$$J = \frac{\theta^T S_b\theta}{\theta^T S_w\theta}$$

我们优化的目标就是使 J 最大化。注意到上式的分子和分母都是关于 θ 的二次项，因此解仅与 θ 的方向有关。不失一般性，令 $\theta^T S_w\theta = 1$，则可得到以下优化目标：

$$\begin{aligned}&\min_\theta -\theta^T S_b\theta \\ &s.t. \theta^T S_w\theta = 1\end{aligned}$$

利用拉格朗日乘子法化解上式可得：

$$c(\theta) = \theta^T S_b \theta - \lambda(\theta^T S_w \theta - 1)$$

$$\frac{dc}{d\theta} = 2S_b\theta - 2\lambda S_w\theta = 0$$

$$S_w^{-1} S_b \theta = \lambda\theta$$

参数 θ 为矩阵 $S_w^{-1}S_b$ 的特征向量。注意到 $S_b\theta$ 的方向恒为 $\mu_0 - \mu_1$，不妨令 $S_b\theta = \lambda(\mu_0 - \mu_1)$。解释：

$$S_b\theta = (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T\theta$$

其中 $(\mu_0 - \mu_1)^T\theta$ 为一个常数。

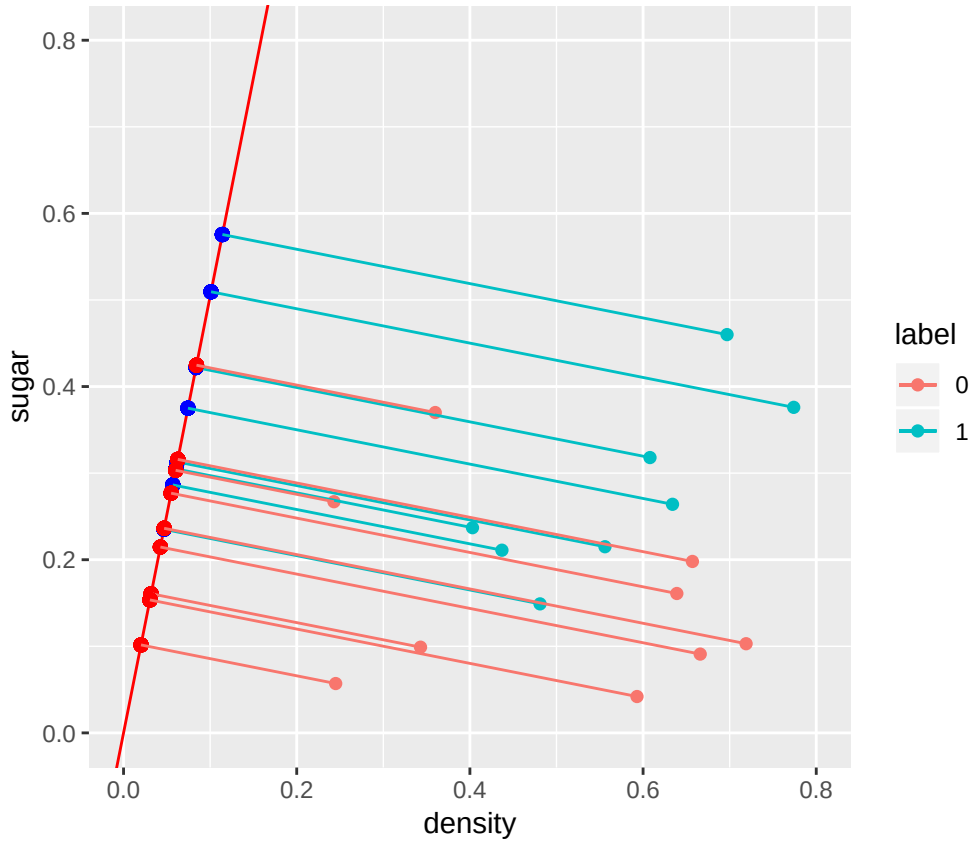
最终化简结果为：

$$\theta = S_w^{-1}(\mu_0 - \mu_1)$$

- 编程实现

```
LDA <- function(X, y){
  pos <- which(y == 1)
  neg <- which(y == 0)
  u1 <- as.matrix(colMeans(X[pos, ]))
  u0 <- as.matrix(colMeans(X[neg, ]))
  Sw <- (t(X[pos, ]) - as.numeric(u1)) %*% t(t(X[pos, ]) - as.numeric(u1)) +
    (t(X[neg, ]) - as.numeric(u0)) %*% t(t(X[neg, ]) - as.numeric(u0))
  theta <- ginv(Sw) %*% (u0 - u1)
  return(list(u1=u1, u0=u0, theta = theta))
}

X <- as.matrix(wmda[, 2:3])
y <- as.matrix(as.numeric(as.character(wmda$label)))
Ret <- LDA(X, y)
theta <- Ret$theta
print(theta)
```

```
##             [,1]
## [1,] -0.1465098
## [2,] -0.7387156
```



- 备注：

θ^T 就是我们得到的直线，在图中展示就是斜率为 `theta[2]/theta[1]`, 截距为 0 的直线。图中的映射线与 θ^T 垂直，我们通过点斜式计算出每一点的映射线，再与直线 θ^T 联立求得每一点在 θ^T 直线上与之相对应的点。设 $k = \theta_2/\theta_1$ 为直线 θ^T 的斜率，则 θ^T 也可以表示为 $y = kx$。映射线斜率为 $-1/k$。下面以一个点为例计算该点在 $y = kx$ 上的投影点。

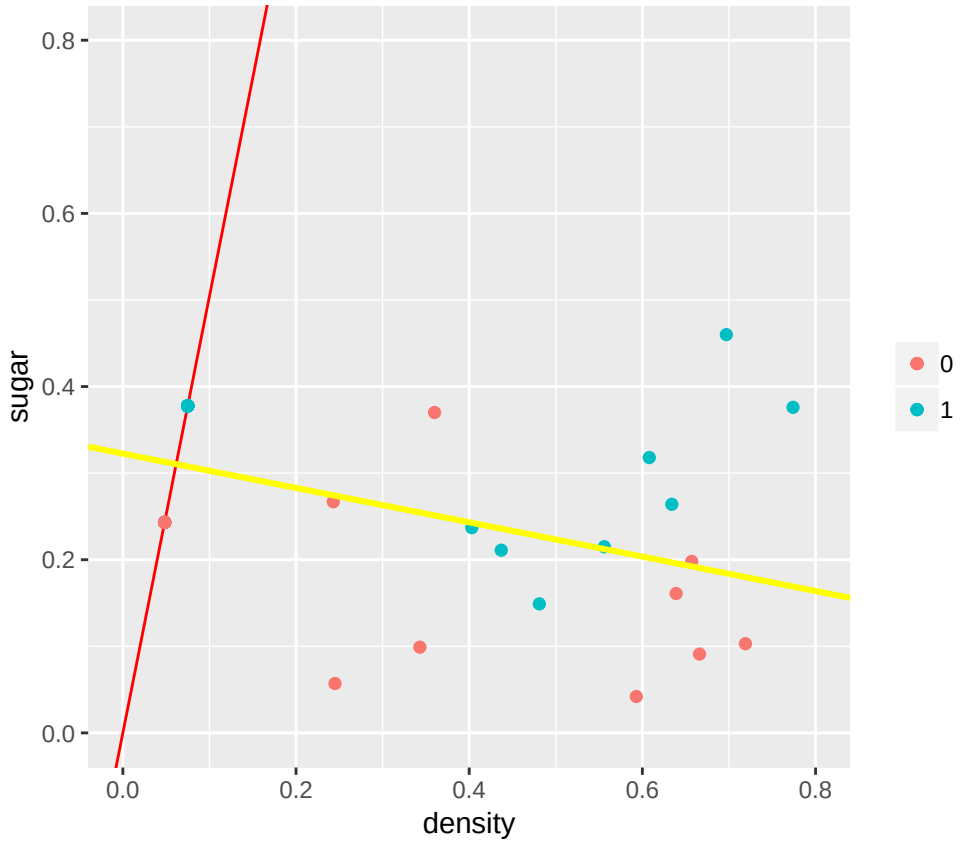
$$\begin{cases} y = kx \\ y - y_i = -1/k(x - x_i) \end{cases}$$

解得

$$x = \frac{\frac{1}{k}x_i + y_i}{\frac{1}{k} + 1}; y = \frac{x_i + ky_i}{\frac{1}{k} + 1}$$

1.3.2 决策边界

在本节开始中有提到对新样本进行分类时，将其投影到同样的这条直线上，再根据投影点的位置来判断新样本的类别。在这里我们取决策边界为 $\theta^T\mu_0$ 与 $\theta^T\mu_1$ 的中心。推导得决策边界的斜率为 $-1/k$，截距为 $\frac{1}{k}x_1 + y_1$。



1.3.3 多分类 LDA

假设存在 N 个类，且第 i 个类的样本数为 m_i。定义“全局散度矩阵”：

$$\begin{aligned} S_t &= S_b + S_w \\ &= \sum_{i=1}^{m}(x_i - \mu)(x_i - \mu)^T \end{aligned}$$

其中 μ 为所有样本的均值向量。S_w 重新定义为：

$$S_w = \sum_{i=1}^{N}\sum_{x \in D_i}(x - \mu_i)(x - \mu_i)^T$$

则 S_b 为：

$$\begin{aligned} S_b &= S_t - S_w \\ &= \sum_{i=1}^{N} m_i(\mu_i - \mu)(\mu_i - \mu)^T \end{aligned}$$

通过求广义特征值方法求解 θ。

$$S_w^{-1} S_b \theta = \lambda\theta$$

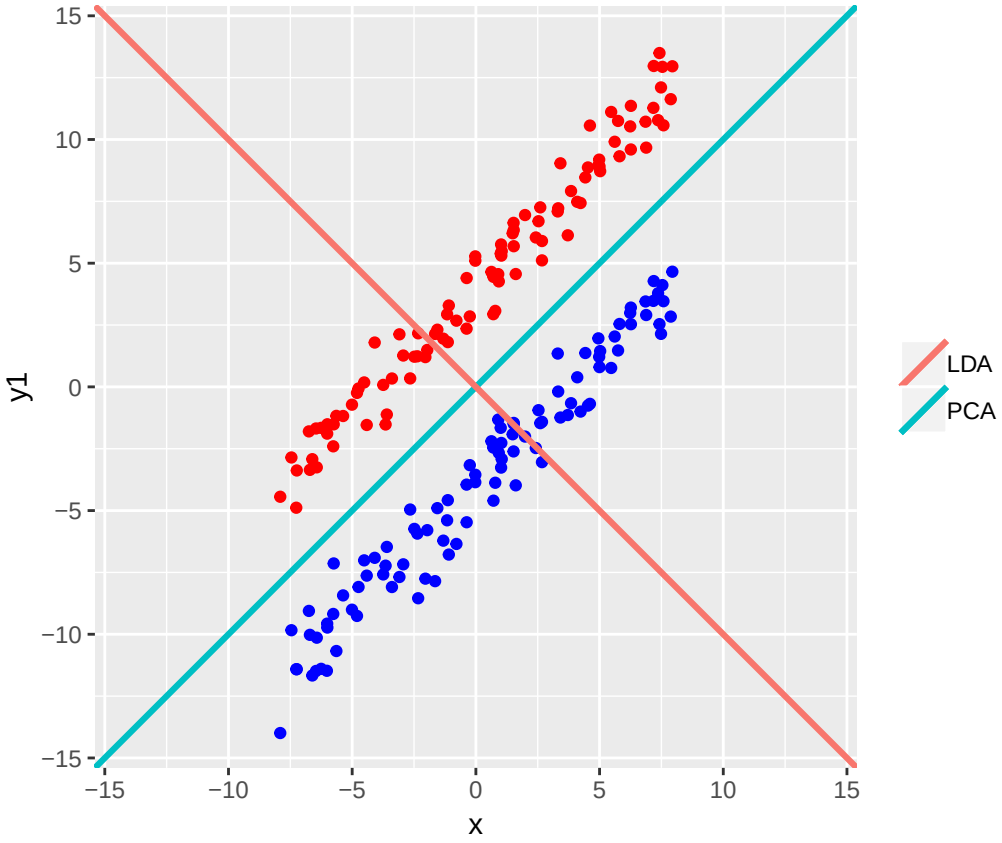
θ 的闭式解是 $S_w^{-1}S_b$ 的 N-1 个最大广义特征值所对应的特征向量组成的矩阵。

1.3.4 小结

- 监督降维

LDA 是一种经典的监督式降维技术。PCA 是一种无监督的数据降维方法。我们知道即使在训练样本上，我们提供了类别标签，在使用 PCA 模型的时候，我们是不利用类别标签的，而 LDA 在进行数据降维的时候是利用数据的类别标签提供的信息的。

从几何的角度来看，PCA 和 LDA 都是将数据投影到新的相互正交的坐标轴上。只不过在投影的过程中他们使用的约束是不同的，也可以说目标是不同的。PCA 是将数据投影到方差最大的几个相互正交的方向上，以期待保留最多的样本信息。样本的方差越大表示样本的多样性越好，在训练模型的时候，我们当然希望数据的差别越大越好。否则即使样本很多但是他们彼此相似或者相同，提供的样本信息将相同，相当于只有很少的样本提供信息是有用的。但是，对于一些特殊分布的数据集，PCA 的这个投影后方差最大的目标就不太合适了。比如下图：



- 奇异值分解

奇异值分解（Singular Value Decomposition, 简称 SVD）是在机器学习领域广泛应用的算法，它不光可以用于降维算法中的特征分解，还可以用于推荐系统，以及自然语言处理等领域。

SVD 是对矩阵进行分解。假设我们的矩阵 A 是一个 $m \times n$ 的矩阵，那么我们定义矩阵 A 的 SVD 为：

$$A = U\Sigma V^T$$

其中 U 是一个 $m \times m$ 的矩阵；Σ 是一个 $m \times n$ 的矩阵，除了主对角线上的元素以外全为 0，主对角线上的元素称为奇异值；V 是一个 $n \times n$ 的矩阵。U 和 V 都是酉矩阵，即满足 $U^TU = I$，$V^TV = I$。下面进行求解。

如果我们将 A 的转置和 A 做矩阵乘法，对 $A^TA(m \times n)$ 进行特征值分解。

$$(A^TA)v_i = \lambda_i v_i$$

这样我们就可以得到矩阵 A^TA 的 n 个特征值和对应的 n 个特征向量 v 了。**将 A^TA 的所有特征向量张成一个 $n \times n$ 的矩阵 V，就是我们 SVD 公式里面的 V 矩阵了，一般我们将 V 中的每一个特征向量叫做 A 的右奇异向量。**

同样我们将 A 和 A 的转置做矩阵乘法，会得到一个 $m \times m$ 的方阵 AA^T。进行特征值分解，然后将所有特征向量张成一个 $m \times m$ 的矩阵 U，这就是我们 SVD 公式里面的 U 矩阵了。一般我们将 U 中的每一个特征向量叫做 A 的左奇异向量。

$$A = U\Sigma V^T \Rightarrow A^T = V\Sigma^T U^T \Rightarrow A^TA = V\Sigma^2 V^T$$

从上式我们可以看出 A^TA 的特征向量张成的矩阵就是我们 SVD 中的 V 矩阵。同时也可以看出 A^TA 的特征值是奇异值的平方，也就是有：

$$\sigma_i = \sqrt{\lambda_i}$$

举例：

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}$$

求得

$$A^TA = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$AA^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

进而求得 A^TA 的特征值与特征向量为：

$$\lambda_1 = 3; v_1 = (\frac{1}{\sqrt{2}}; \frac{1}{\sqrt{2}}); \lambda_2 = 1; v_2 = (-\frac{1}{\sqrt{2}}; \frac{1}{\sqrt{2}})$$

AA^T 的特征值与特征向量为：

$$\lambda_1 = 3; u_1 = (\frac{1}{\sqrt{6}}; \frac{2}{\sqrt{6}}; \frac{1}{\sqrt{6}}); \lambda_2 = 1; u_2 = (\frac{1}{\sqrt{2}}; 0; -\frac{1}{\sqrt{2}}); \lambda_3 = 0; u_3 = (\frac{1}{\sqrt{3}}; -\frac{1}{\sqrt{3}}; \frac{1}{\sqrt{3}})$$

利用 $\sigma_i = \sqrt{\lambda_i}$ 解得，$\sigma_1 = \sqrt{3}, \sigma_2 = 1$。

最终 A 的奇异值分解为：

$$A = U\Sigma V^T = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

备注：`eigen()` 可以用来计算特征值与特征向量。

对于奇异值, 它跟我们特征分解中的特征值类似，在奇异值矩阵中也是按照从大到小排列，而且奇异值的减少特别的快，在很多情况下，前 10% 甚至 1% 的奇异值的和就占了全部的奇异值之和的 99% 以上的比例。也就是说，我们也可以用最大的 k 个的奇异值和对应的左右奇异向量来近似描述矩阵。