Lab: 逻辑回归

李文东

最后编译于 10/15/2022

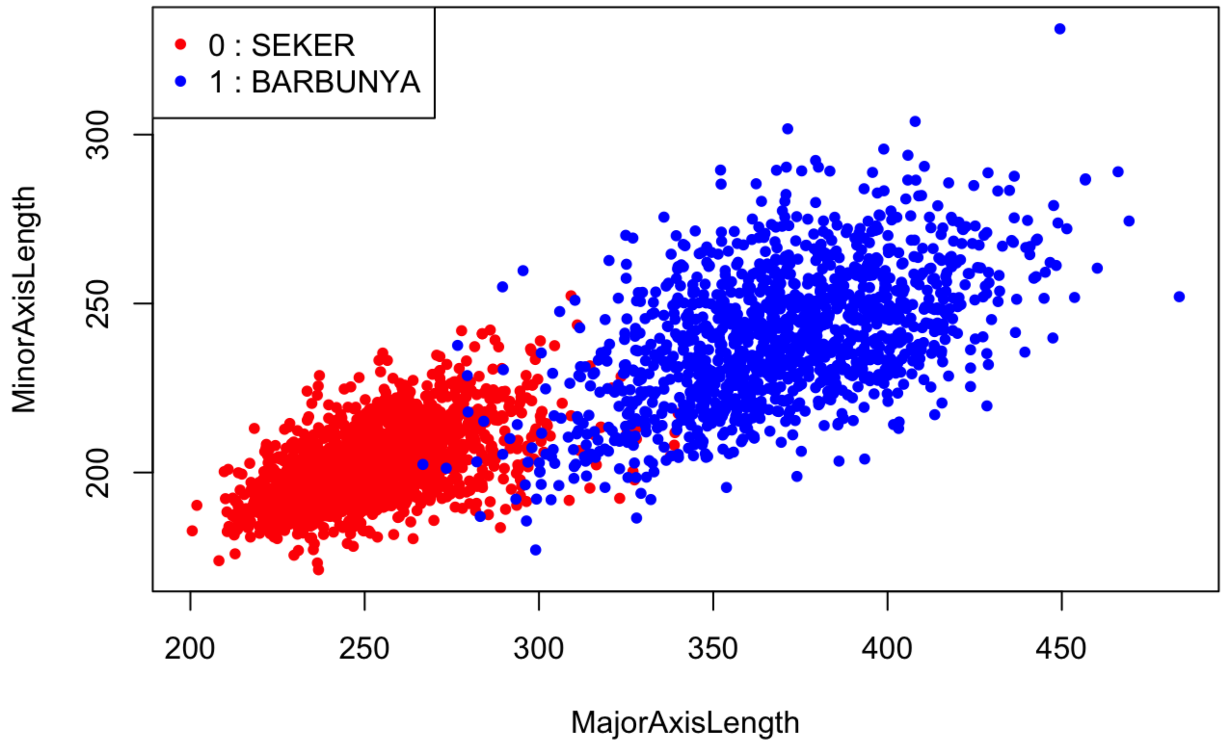
1 数据读取与可视化

本Lab的代码演示主要以Dry_Bean_Dataset (https://www.kaggle.com/datasets/muratkokludataset/dry-bean-dataset)数据集为例。使用高分辨率相机拍摄了7种不同风干菜豆的共13611粒的图像，共有16个特征，其中包括12种尺寸和4种形状特征（菜豆的区域，周长，长轴长、短轴长等等）。 `Class` 代表菜豆的种类。

```
Bean <- read.csv(file = "Dry_Bean_Dataset.csv", header = TRUE)
dim(Bean)
## [1] 13611    17

head(Bean)
##    Area Perimeter MajorAxisLength MinorAxisLength AspectRation Eccentricity
## 1 28395   610.291        208.1781        173.8887     1.197191    0.5498122
## 2 28734   638.018        200.5248        182.7344     1.097356    0.4117853
## 3 29380   624.110        212.8261        175.9311     1.209713    0.5627273
## 4 30008   645.884        210.5580        182.5165     1.153638    0.4986160
## 5 30140   620.134        201.8479        190.2793     1.060798    0.3336797
## 6 30279   634.927        212.5606        181.5102     1.171067    0.5204007
##   ConvexArea EquivDiameter    Extent  Solidity roundness Compactness
## 1      28715      190.1411 0.7639225 0.9888560 0.9580271   0.9133578
## 2      29172      191.2728 0.7839681 0.9849856 0.8870336   0.9538608
## 3      29690      193.4109 0.7781132 0.9895588 0.9478495   0.9087742
## 4      30724      195.4671 0.7826813 0.9766957 0.9039364   0.9283288
## 5      30417      195.8965 0.7730980 0.9908932 0.9848771   0.9705155
## 6      30600      196.3477 0.7756885 0.9895098 0.9438518   0.9237260
##   ShapeFactor1 ShapeFactor2 ShapeFactor3 ShapeFactor4 Class
## 1  0.007331506  0.003147289    0.8342224    0.9987239 SEKER
## 2  0.006978659  0.003563624    0.9098505    0.9984303 SEKER
## 3  0.007243912  0.003047733    0.8258706    0.9990661 SEKER
## 4  0.007016729  0.003214562    0.8617944    0.9941988 SEKER
## 5  0.006697010  0.003664972    0.9419004    0.9991661 SEKER
## 6  0.007020065  0.003152779    0.8532696    0.9992358 SEKER
```

为了便于在二维平面可视化，我们选取 `MajorAxisLength` 和 `MinorAxisLength` 两个特征。由于是两分类，我们选取 `SEKER` 和 `BARBUNYA` 这两个种类，即前3349组数据。绘制散点图观察。

```
Bean.Binary <- Bean[1:3349,c("MajorAxisLength","MinorAxisLength","Class")]
Bean.Binary[Bean.Binary[,3]=="SEKER",3]=0
Bean.Binary[Bean.Binary[,3]=="BARBUNYA",3]=1
Bean.Binary[,3] <- as.numeric(Bean.Binary[,3])
n1 <- sum(Bean.Binary[,3]==0); n2 <- sum(Bean.Binary[,3]==1)
c(n1,n2)
## [1] 2027 1322

plot(Bean.Binary[,1],Bean.Binary[,2],col=c(rep("red",n1),rep("blue",n2)),xlab = "MajorAxisLength",ylab = "MinorAxisLength",pch=20)
legend("topleft",legend=c("0 : SEKER","1 : BARBUNYA"),col=c("red","blue"),pch=20)
```



2 基于glm()函数实现逻辑回归

下面我们拟合逻辑回归模型来基于 `MajorAxisLength` 和 `MinorAxisLength` 对菜豆的 `Class` 进行预测。 `glm()` 函数可以被用来拟合很多种类的广义线性模型（Generalized Linear Model），其中就包括逻辑回归。 `glm()` 的用法与 `lm()` 几乎一致，除了我们要输入参数 `family = binomial` 来告诉 `R` 去运行逻辑回归而不是其他的广义线性模型。

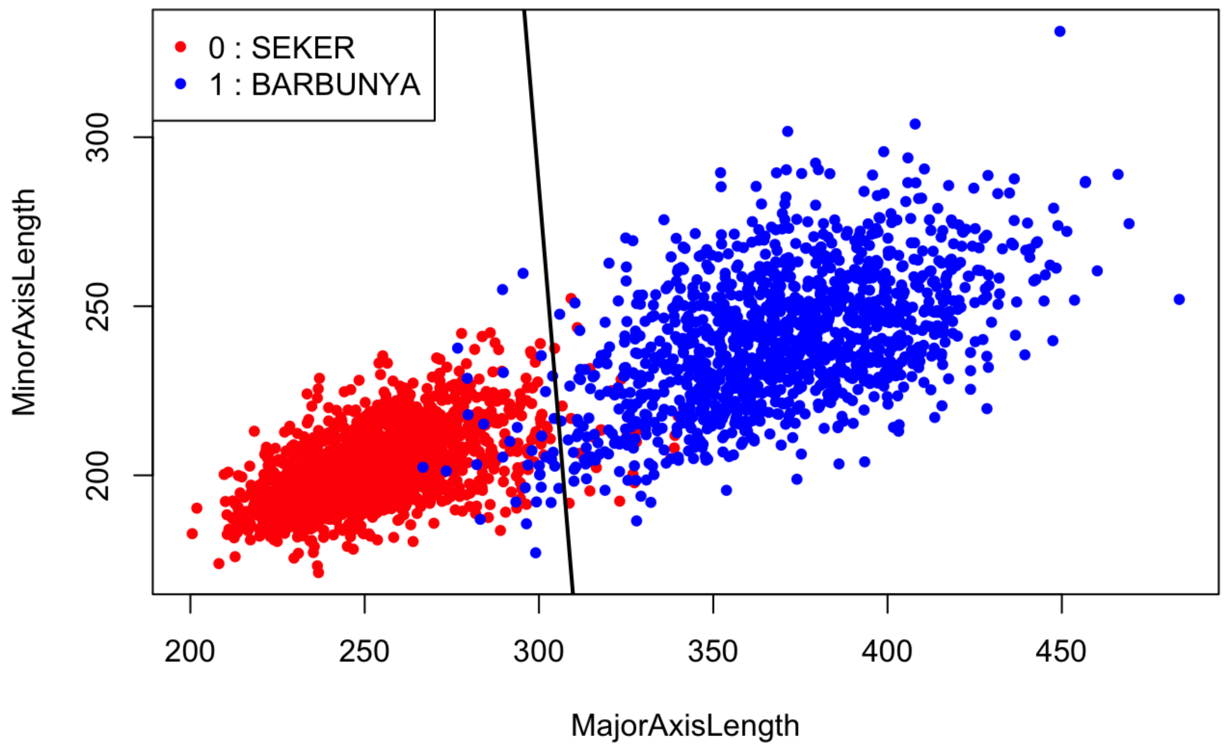
```
glm.fits <- glm(Class ~ MajorAxisLength + MinorAxisLength, data=Bean.Binary, family = binomial)
summary(glm.fits)
```

```
##
## Call:
## glm(formula = Class ~ MajorAxisLength + MinorAxisLength, family = binomial,
##     data = Bean.Binary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.92175  -0.05136  -0.01453   0.01208   3.14521
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -39.897670   2.695966 -14.799   <2e-16 ***
## MajorAxisLength   0.123471   0.007747  15.938   <2e-16 ***
## MinorAxisLength   0.010024   0.009909   1.012    0.312
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4493.17  on 3348  degrees of freedom
## Residual deviance:  340.76  on 3346  degrees of freedom
## AIC: 346.76
##
## Number of Fisher Scoring iterations: 9
```

```
coef(glm.fits)
```

```
##     (Intercept) MajorAxisLength MinorAxisLength
##    -39.89766997      0.12347146      0.01002436
```

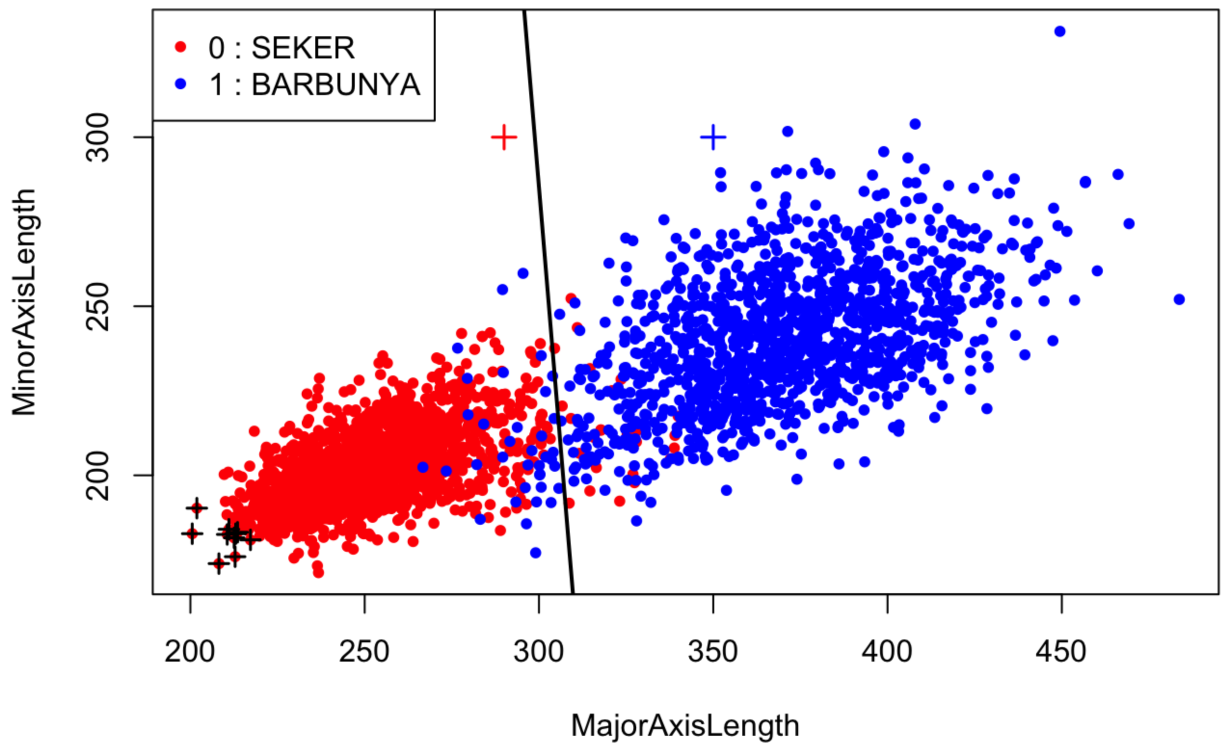
```
plot(Bean.Binary[,1],Bean.Binary[,2],col=c(rep("red",n1),rep("blue",n2)),xlab = "MajorAxisLength",ylab = "MinorAxisLength",pch=20)
legend("topleft",legend=c("0 : SEKER","1 : BARBUNYA"),col=c("red","blue"),pch=20)
abline(-coef(glm.fits)[1]/coef(glm.fits)[3],-coef(glm.fits)[2]/coef(glm.fits)[3],lwd=2)
```



`predict()` 函数可以被用来在给定特征值的情况下预测菜豆属于 BARBUNYA 的概率。参数 `type = "response"` 告诉 R 来以$P(Y=1|X)$的形式输出概率。如果没有新的数据集输入到 `predict()` 当中，函数会默认输出训练数据集对应的概率值。

下面我们打印了前十个训练数据的概率预测值，并给出了它们对应的分类预测值。从下图中我们也可以发现这十个观测离分类边界非常远，这是它们的概率值都很极端的原因。

```
glm.probs <- predict(glm.fits, type = "response")
glm.probs[1:10]
##            1            2            3            4            5            6
## 3.915789e-06 1.663169e-06 7.094959e-06 5.727909e-06 2.112179e-06 7.261017e-06
##            7            8            9           10
## 6.180380e-06 7.757635e-06 8.324800e-06 1.284103e-05

glm.labels <- as.numeric(glm.probs>=0.5)
glm.labels[1:10]
##  [1] 0 0 0 0 0 0 0 0 0 0

predict(glm.fits, newdata = data.frame(MajorAxisLength = c(290, 350), MinorAxisLength = c
(300, 300)), type = "response")
##         1         2
## 0.2528183 0.9982116

plot(Bean.Binary[,1],Bean.Binary[,2],col=c(rep("red",n1),rep("blue",n2)),xlab = "MajorAxi
sLength",ylab = "MinorAxisLength",pch=20)
legend("topleft",legend=c("0 : SEKER","1 : BARBUNYA"),col=c("red","blue"), pch=20)
abline(-coef(glm.fits)[1]/coef(glm.fits)[3],-coef(glm.fits)[2]/coef(glm.fits)[3], lwd=2)
points(Bean.Binary[1:10,1],Bean.Binary[1:10,2], pch=3, col="black", lwd=1.5)
points(c(290, 350),c(300, 300), pch=3, col=c("red","blue"), cex=1.2, lwd=1.5)
```



基于这些预测，我们可以利用 `table()` 函数来产生混淆矩阵(confusion matrix)来判断有多少观测被正确或错误分类。通过向 `table()` 函数输入两个向量， `R` 会输出一个二乘二的表格，包括了各种情况下的计数。混淆矩阵的对角元素代表着正确的预测，非对角元素代表着错误的预测。因此我们的模型正确预测了2000个SEKER菜豆和1286个BARBUNYA菜豆。 `mean()` 函数可以用来计算预测准确率。在这个例子中，逻辑回归准确的预测了98.12%的菜豆。

```
True.labels <- Bean.Binary[,3]
table(glm.labels, True.labels)
##             True.labels
## glm.labels     0    1
##          0  2000   36
##          1    27 1286
(2000+1286)/3349
## [1] 0.9811884
mean(glm.labels == True.labels)
## [1] 0.9811884
```

值得注意的是，上面计算的是训练误差(training error)，所以某种程度上来说，并不代表着我们的模型的泛化能力强。如何计算测试误差我们将在后续的章节中介绍。

3 梯度下降算法实现逻辑回归

我们首先定义损失函数，方便调用。

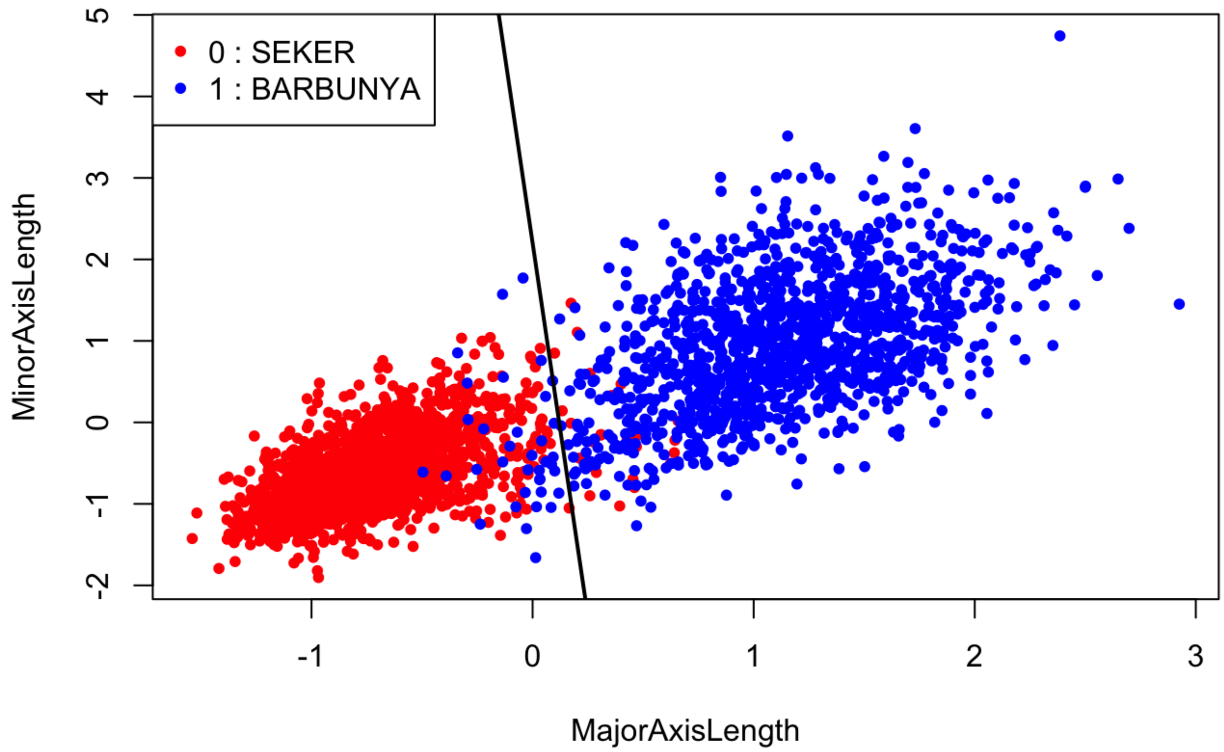
```
cost <- function(x,y,beta){
  sig <- 1/(1+exp(-x %*% beta))
  return(-mean(y*log(sig)+(1-y)*log(1-sig)))
}
```

接下来我们编写梯度下降算法来实现线性回归。

```
alpha <- 0.05   #learning rate

x <- cbind(1,Bean.Binary[,1],Bean.Binary[,2])
x[,2:3] <- scale(x[,2:3])
y <- Bean.Binary[,3]
beta <- as.matrix(c(0,0.1,-0.15),ncol=1)
cost.history <- cost(x,y,beta)

tem.beta1 <- beta[1]+alpha*mean(y-1/(1+exp(-x %*% beta)))
tem.beta2 <- beta[2]+alpha*mean((y-1/(1+exp(-x %*% beta)))*x[,2])
tem.beta3 <- beta[3]+alpha*mean((y-1/(1+exp(-x %*% beta)))*x[,3])
beta[1] <- tem.beta1; beta[2] <- tem.beta2; beta[3] <- tem.beta3;
cost.history <- c(cost.history,cost(x,y,beta))

for (i in 1:20000) {
  tem.beta1 <- beta[1]+alpha*mean(y-1/(1+exp(-x %*% beta)))
  tem.beta2 <- beta[2]+alpha*mean((y-1/(1+exp(-x %*% beta)))*x[,2])
  tem.beta3 <- beta[3]+alpha*mean((y-1/(1+exp(-x %*% beta)))*x[,3])
  beta[1] <- tem.beta1; beta[2] <- tem.beta2; beta[3] <- tem.beta3;
  cost.history <- c(cost.history,cost(x,y,beta))
}
```

下面我们来看一些迭代结束后的结果：

```
beta  #最终参数
##            [,1]
## [1,] -0.8511256
## [2,]  7.0882817
## [3,]  0.3875887
cost.history[length(cost.history)] #最终损失函数
## [1] 0.05126197
plot(x[,2],x[,3],col=c(rep("red",n1),rep("blue",n2)),xlab = "MajorAxisLength",ylab = "Min
orAxisLength",pch=20)
legend("topleft",legend=c("0 : SEKER","1 : BARBUNYA"),col=c("red","blue"), pch=20)
abline(-beta[1]/beta[3],-beta[2]/beta[3], lwd=2)
```




```
plot(cost.history)
```