1 数据读取与可视化

2 Ridge Regression & LASSO

# Lab: 正则化

李文东

最后编译于 11/06/2022

# 1 数据读取与可视化

本lab的数据分析主要基于 `Hitters` 数据集。我们想要基于一个棒球运动员在上一年的各项数据统计来预测其工资 `Salary` 。

首先，有些球员的 `Salary` 变量是缺失的。我们可以利用 `is.na()` 函数来识别这些缺失值。此函数会返回一个相同长度的向量，其中元素为 `TRUE` 代表缺失， `FALSE` 代表未缺失。结合 `sum()` 函数则可以统计有多少个缺失值。

```
library(ISLR2)
names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
##  [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
## [19] "Salary"    "NewLeague"
```

```
dim(Hitters)
```

```
## [1] 322  20
```

```
sum(is.na(Hitters$Salary))
```

```
## [1] 59
```

这里我们看到 `Salary` 有59个缺失值。 `na.omit()` 函数可以自动删除带有任意缺失值的行。

```
Hitters <- na.omit(Hitters)
dim(Hitters)
```

```
## [1] 263  20
```

```
sum(is.na(Hitters))
```

```
## [1] 0
```

# 2 Ridge Regression & LASSO

我们将使用 `glmnet` 包来实现岭回归和LASSO，其中的主要函数是 `glmnet()` ，可以被用来拟合包括岭回归、LASSO等在内的很多模型。

这个函数和我们之前学习过的很多模型拟合类函数（如 `lm()` 和 `glm()` ）的调用方式有所不同。特别的，我们必须输入一个 `x` 矩阵和 `y` 向量，并不再使用以前的{}形式。

我们现在将利用岭回归和LASSO来预测球员的工资。在开始之前，请一定确保所有的缺失值都已被移除。

```
x <- model.matrix(Salary ~ ., data = Hitters)[, -1]
y <- Hitters$Salary
head(x)
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Alan Ashby         315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis        479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga   321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin    594  169     4   74  51    35    11   4408  1133     19
## -Al Newman          185   37     1   23   8    21     2    214    42      1
##                   CRuns CRBI CWalks LeagueN DivisionW PutOuts Assists Errors
## -Alan Ashby         321  414    375       1         1     632      43     10
## -Alvin Davis        224  266    263       0         1     880      82     14
## -Andre Dawson       828  838    354       1         0     200      11      3
## -Andres Galarraga    48   46     33       1         0     805      40      4
## -Alfredo Griffin    501  336    194       0         1     282     421     25
## -Al Newman           30    9     24       1         0      76     127      7
##                   NewLeagueN
## -Alan Ashby                1
## -Alvin Davis               0
## -Andre Dawson              1
## -Andres Galarraga          1
## -Alfredo Griffin           0
## -Al Newman                 0
```

`model.matrix()` 函数对于创造设计矩阵 `x` 非常有用。一方面，它可以创造一个矩阵包含了所有的19个特征；另一方面，它会将所有的定性变量自动转化为虚拟变量。第二点非常重要，因为 `glmnet()` 只接收数值型定量变量的输入。

# 2.1 Ridge Regression

`glmnet()` 函数中的 `alpha` 参数决定了所使用的是何种模型。如果 `alpha=0` ，模型为岭回归；如果 `alpha=1` ，模型为LASSO。我们首先拟合岭回归模型。

默认情况下， `glmnet()` 函数会在自动选择的一系列$\lambda$下运行岭回归。在这里我们选择从$\lambda = 10^{10}$到$\lambda = 10^{-2}$，基本上包括了所有情况，从只包含截距项，到传统的最小二乘。

注意在默认情况下， `glmnet()` 会对变量做标准化使得它们都在同样的维度上。若想要关闭这一设定，令 `standardize = FALSE` 。

```
library(glmnet)
grid <- 10^seq(10, -2, length = 100)
ridge.mod <- glmnet(x, y, alpha = 0, lambda = grid)
```

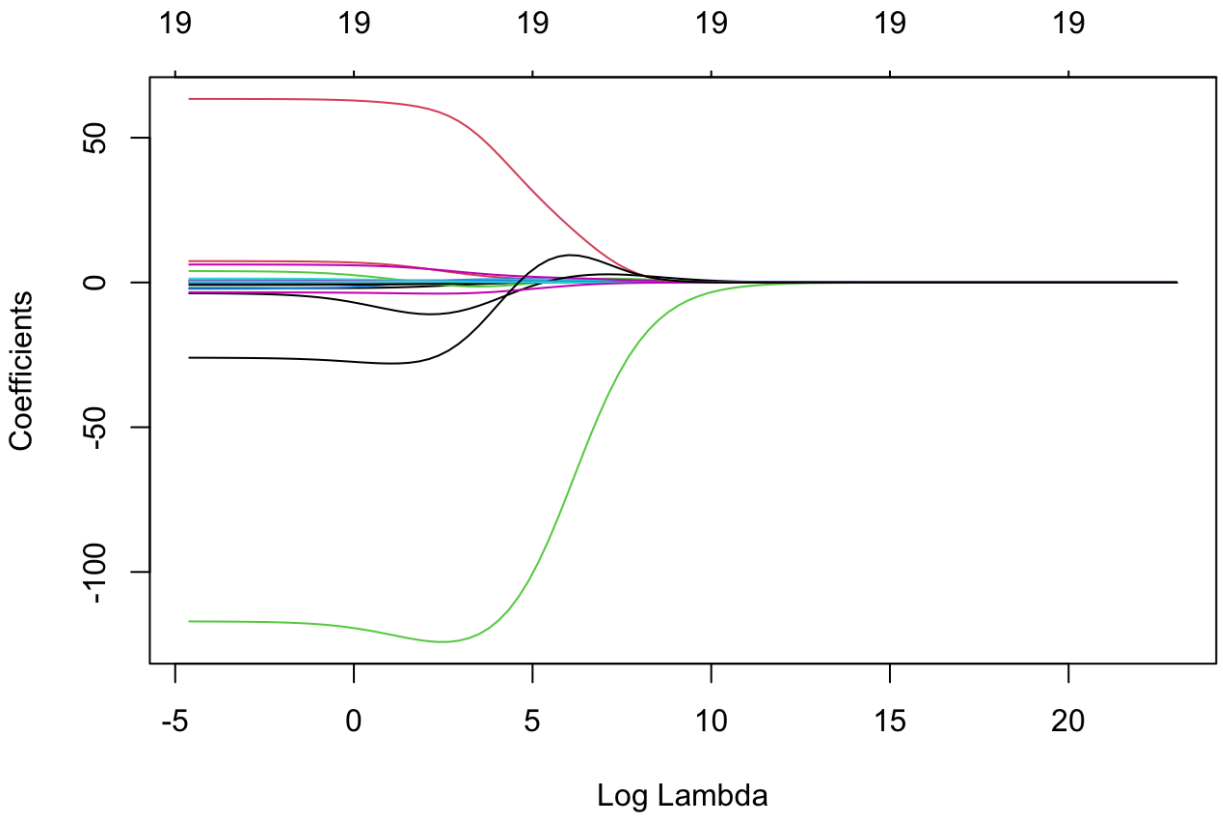
给定每一个$\lambda$的值，我们都能得到一个岭回归系数向量，这些值合并保存在一个矩阵中，可以通过`coef()`获取。在本例中，这是一个$20 \times 100$的矩阵，每一行代表一个特征（包括截距项），每一列代表一个$\lambda$值。

```
dim(coef(ridge.mod))
```

```
## [1]  20 100
```

我们可以利用`plot()`函数来对系数进行可视化。每一条线代表一个特征，它显示了当$\lambda$变化时其系数的变化路径。最上面一排的数字代表当前模型中的特征个数。

```
plot(ridge.mod, xvar = "lambda")
```




我们预计，当使用较大的$\lambda$值时，与使用较小的$\lambda$值相比，从L2范数角度来说，系数的估计是要更小的。下面展示的是当$\lambda = 11{,}498$时的参数，以及其L2范数：

```
ridge.mod$lambda[50]
```

```
## [1] 11497.57
```

```
coef(ridge.mod)[, 50]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
## 407.356050200   0.036957182   0.138180344   0.524629976   0.230701523
##           RBI         Walks         Years        CAtBat         CHits
##   0.239841459   0.289618741   1.107702929   0.003131815   0.011653637
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##   0.087545670   0.023379882   0.024138320   0.025015421   0.085028114
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
##  -6.215440973   0.016482577   0.002612988  -0.020502690   0.301433531
```

```
sqrt(sum(coef(ridge.mod)[-1, 50]^2))
```

```
## [1] 6.360612
```

作为对比，下面展示的是当$\lambda = 705$时的参数，以及其L2范数：

```
ridge.mod$lambda[60]
```

```
## [1] 705.4802
```

```
coef(ridge.mod)[, 60]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  54.32519950   0.11211115   0.65622409   1.17980910   0.93769713   0.84718546
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##   1.31987948   2.59640425   0.01083413   0.04674557   0.33777318   0.09355528
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   0.09780402   0.07189612  13.68370191 -54.65877750   0.11852289   0.01606037
##       Errors   NewLeagueN
##  -0.70358655   8.61181213
```

```
sqrt(sum(coef(ridge.mod)[-1, 60]^2))
```

```
## [1] 57.11001
```

我们可以利用 `predict()` 函数实现很多目的。

对于一个没有出现过的$\lambda$值（例如50），我们同样可以获得其岭回归系数：

```
predict(ridge.mod, s = 50, type = "coefficients")
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  4.876610e+01
## AtBat       -3.580999e-01
## Hits         1.969359e+00
## HmRun       -1.278248e+00
## Runs         1.145892e+00
## RBI          8.038292e-01
## Walks        2.716186e+00
## Years       -6.218319e+00
## CAtBat       5.447837e-03
## CHits        1.064895e-01
## CHmRun       6.244860e-01
## CRuns        2.214985e-01
## CRBI         2.186914e-01
## CWalks      -1.500245e-01
## LeagueN      4.592589e+01
## DivisionW   -1.182011e+02
## PutOuts      2.502322e-01
## Assists      1.215665e-01
## Errors      -3.278600e+00
## NewLeagueN  -9.496680e+00
```

我们现在将样本分成训练集和测试集，以估计岭回归和LASSO的测试误差。 有两种常见的方法来随机拆分数据集。第一种是产生一个由 TRUE 、 FALSE 元素组成的随机向量，并为训练数据选择与 TRUE 相对应的观测值。

第二种是在1和$n$之间随机选择一个数字子集，然后可以将这些用作训练数据的指标。这两种方法同样有效。

我们基于第二种进行展示，大家可以尝试第一种方法，利用二项分布控制 TRUE 出现的概率。

```
set.seed(1)
train <- sample(1:nrow(x), nrow(x) / 2)
test <- (-train)
y.test <- y[test]
```

```
"展示区域"
```

```
## [1] "展示区域"
```

接下来我们给定$\lambda = 4$，对于训练数据集拟合岭回归模型，并在测试集上计算其MSE。这次我们还是利用 predict() 函数，更改为 type="response" 并增加 newx 参数。

```
ridge.mod <- glmnet(x[train, ], y[train], alpha = 0, lambda = grid)
ridge.pred <- predict(ridge.mod, s = 4, type="response", newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
## [1] 142226.5
```

测试MSE为142226。

注意，如果我们只是简单地拟合一个只有截距的模型，我们将使用训练数据的平均值来预测每个测试数据。在这种情况下，我们可以像这样计算测试集的 MSE：

```
mean((mean(y[train]) - y.test)^2)
```

```
## [1] 224669.9
```

我们也可以通过设置一个非常大的λ来得到同样的结果：

```
ridge.pred <- predict(ridge.mod, s = 1e10, newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
## [1] 224669.8
```

可以看到，$\lambda = 4$时的岭回归比只有截距项的模型有着更小的MSE。 我们再来检查一下岭回归是否优于最小二乘。回顾一下，最小二乘就是$\lambda = 0$时的岭回归。

为了使 `glmnet()` 函数在$\lambda = 0$时产生精确的最小二乘估计系数，我们在调用 `predict()` 函数时需要设置 `exact = TRUE` 。否则， `predict()` 函数会在用来拟合 `glmnet()` 模型的λ的基础上进行差值，从而只能得到近似的结果。

值得注意的是，当我们使用 `exact = T` ，$\lambda = 0$时 `glmnet()` 的输出和 `lm()` 相比，小数点后第三位仍有细微差异。这是因为 `glmnet()` 使用了基于数值近似的迭代算法。

```
ridge.pred <- predict(ridge.mod, s = 0, newx = x[test, ],
    exact = TRUE, x = x[train, ], y = y[train])
mean((ridge.pred - y.test)^2)
```

```
## [1] 167018.2
```

```
predict(ridge.mod, s = 0, exact = TRUE, type = "coefficients",
    x = x[train, ], y = y[train])
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  275.5663772
## AtBat         -0.3944133
## Hits          -1.4485776
## HmRun          5.9964246
## Runs           1.4915557
## RBI            1.0348026
## Walks          3.7792456
## Years        -17.0527764
## CAtBat        -0.6181138
## CHits          3.0646427
## CHmRun         3.2613558
## CRuns         -0.9573405
## CRBI          -0.5403284
## CWalks         0.3282663
## LeagueN      117.2463136
## DivisionW   -144.8421995
## PutOuts        0.1964112
## Assists        0.6750900
## Errors        -4.6935735
## NewLeagueN   -68.9411272
```

**小练习**：对训练数据集拟合 `lm()` ，观察其系数和 `glmnet()` 是否相同。

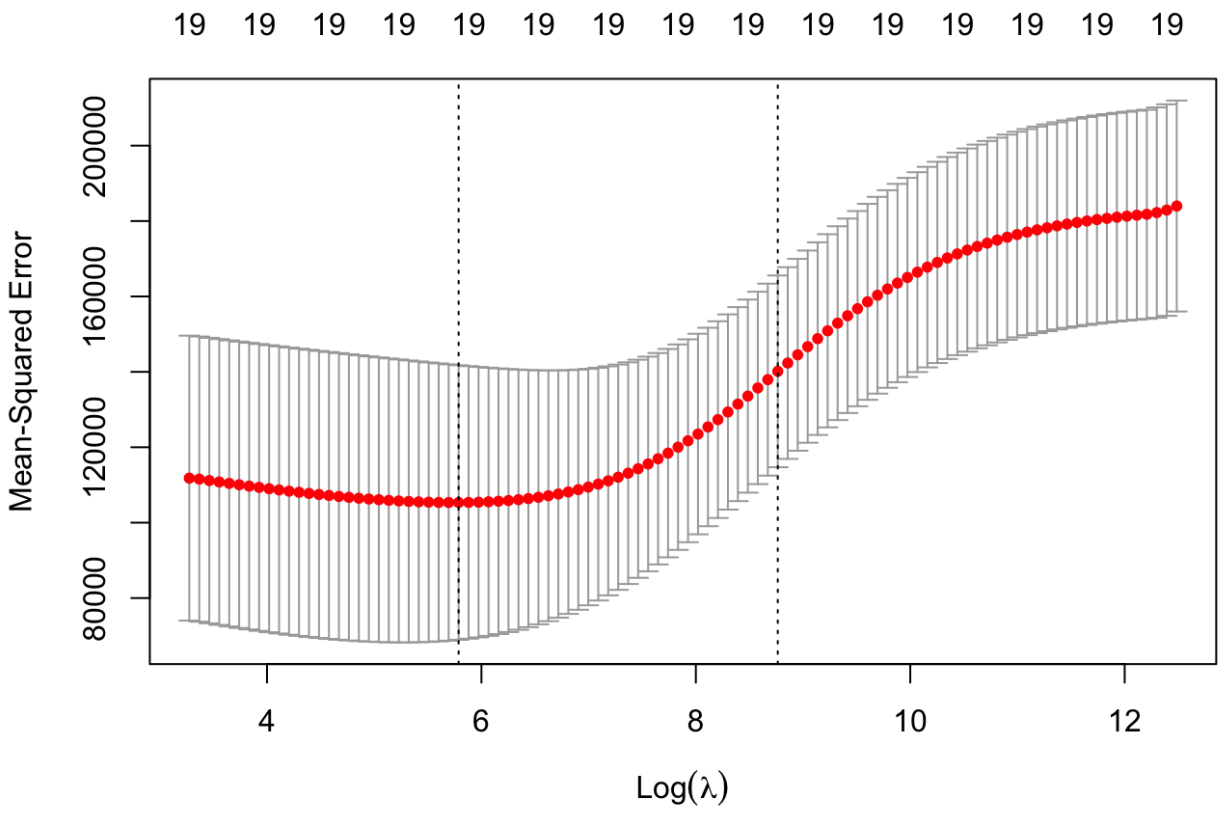
```
"展示区域"
```

```
## [1] "展示区域"
```

一般来说，如果我们要拟合一个不加惩罚的最小二乘模型，我们应该直接使用 `lm()` 函数，因为这个函数提供了更多有用的输出，例如显著性等等。

一般来说，我们不应该武断的选择$\lambda = 4$，而是应该使用cross-validation来选择$\lambda$。这可以通过 `cv.glmnet()` 函数实现。

```
set.seed(1)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 0)
plot(cv.out)
```




```
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 326.0828
```

我们看到有着最小cross-validation误差的λ为326。那么与之对应的测试MSE是多少呢?

```
ridge.pred <- predict(ridge.mod, s = bestlam, newx = x[test, ])
mean((ridge.pred - y.test)^2)
```

```
## [1] 139833.6
```

这比$\lambda = 4$的结果又有了进一步的改进。最后我们用全部的数据集来拟合岭回归。和预想中一样，所有的系数都是非零的，岭回归并没有变量选择的作用。

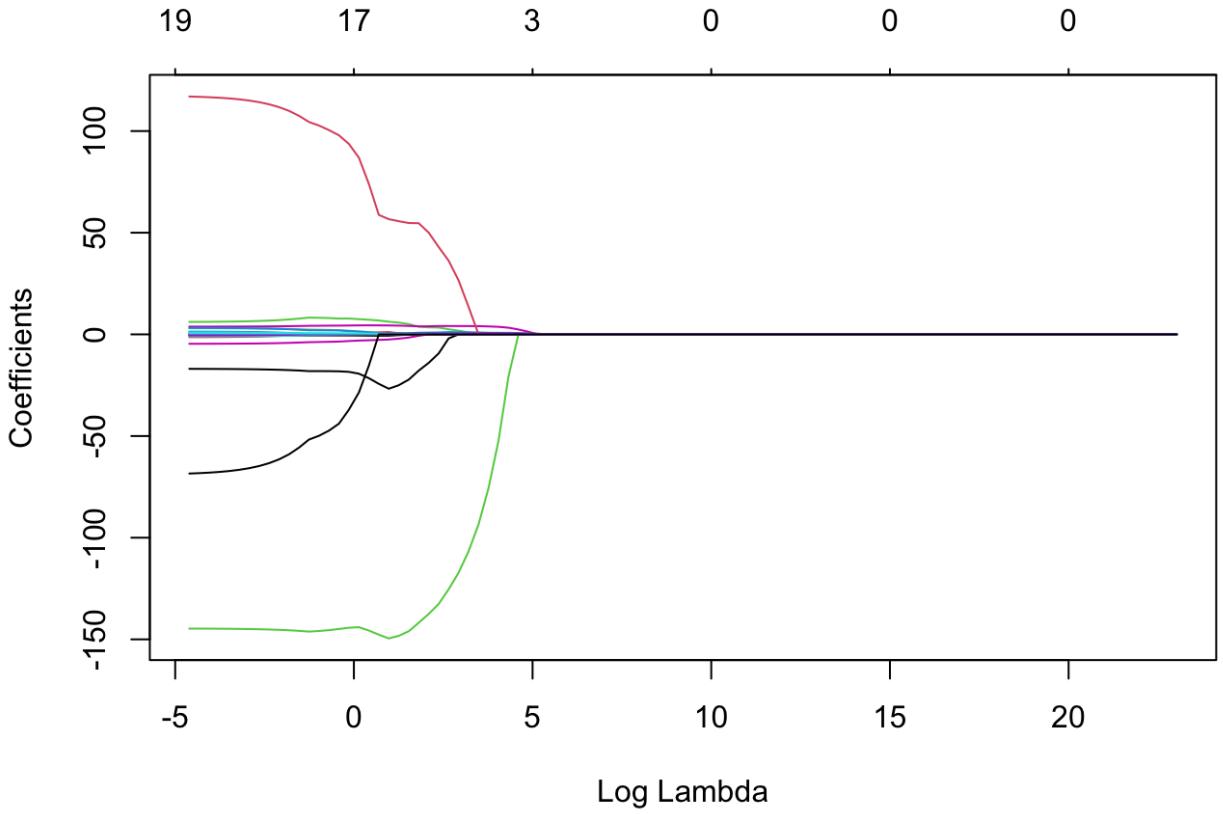
```
out <- glmnet(x, y, alpha = 0)
predict(out, type = "coefficients", s = bestlam)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  15.44383120
## AtBat         0.07715547
## Hits          0.85911582
## HmRun         0.60103106
## Runs          1.06369007
## RBI           0.87936105
## Walks         1.62444617
## Years         1.35254778
## CAtBat        0.01134999
## CHits         0.05746654
## CHmRun        0.40680157
## CRuns         0.11456224
## CRBI          0.12116504
## CWalks        0.05299202
## LeagueN      22.09143197
## DivisionW   -79.04032656
## PutOuts       0.16619903
## Assists       0.02941950
## Errors       -1.36092945
## NewLeagueN    9.12487765
```

# 2.2 LASSO

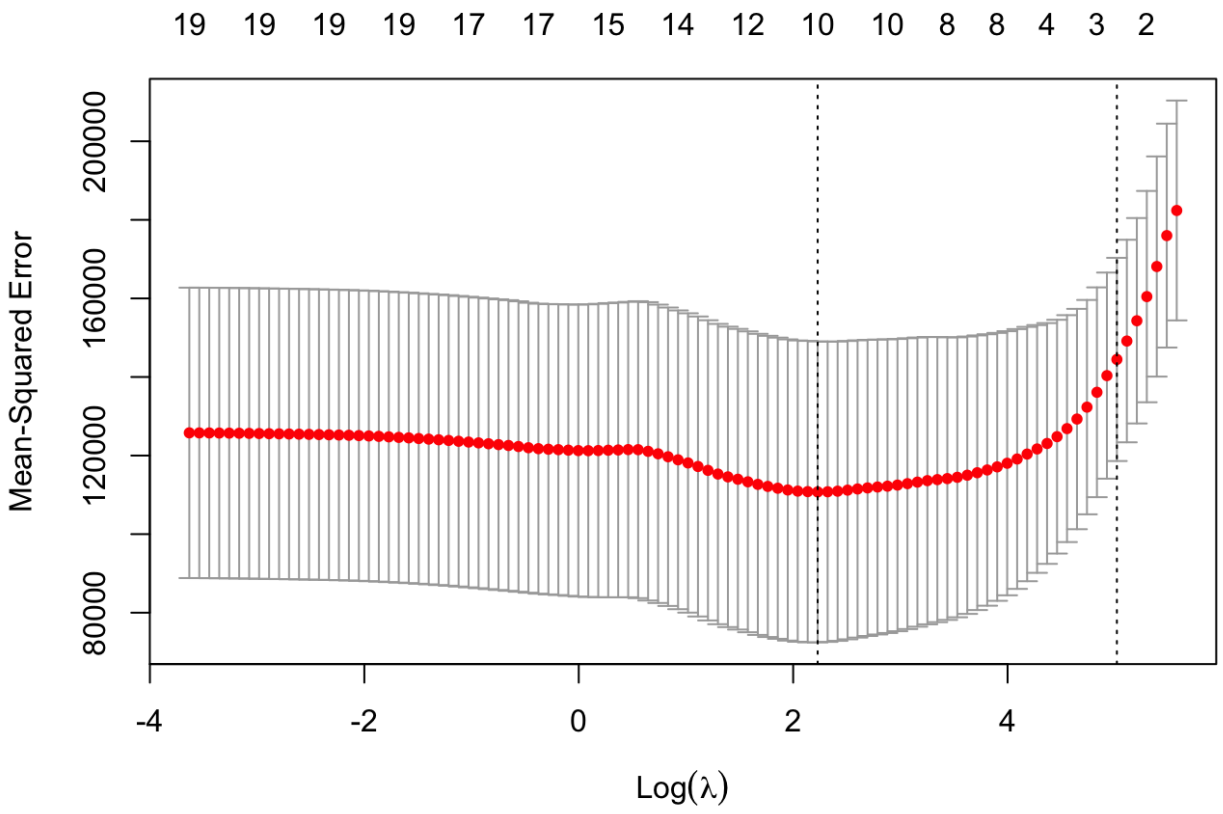
为了拟合LASSO模型，我们还是使用 `glmnet()` 函数，只不过将参数调整为 `alpha=1` 。

```
lasso.mod <- glmnet(x[train, ], y[train], alpha = 1, lambda = grid)
plot(lasso.mod, xvar = "lambda")
```




我们可以看到随着λ的增大，一些系数变成了严格等于0。类似的，我们可以利用cross-validation来计算MSE。

```
set.seed(1)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv.out)
```




```
bestlam <- cv.out$lambda.min
lasso.pred <- predict(lasso.mod, s = bestlam, newx = x[test, ])
mean((lasso.pred - y.test)^2)
```

```
## [1] 143673.6
```

这个结果和岭回归比较类似。然而，LASSO有着岭回归不具有的巨大优势：最终的系数估计是稀疏的。我们看到19个特征中有8个特征的系数等于0，最终的模型中仅包含11个特征。这大大加强了模型的可解释性。

```
out <- glmnet(x, y, alpha = 1, lambda = grid)
predict(out, type = "coefficients", s = bestlam)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)    1.27479059
## AtBat         -0.05497143
## Hits           2.18034583
## HmRun          .
## Runs           .
## RBI            .
## Walks          2.29192406
## Years         -0.33806109
## CAtBat         .
## CHits          .
## CHmRun         0.02825013
## CRuns          0.21628385
## CRBI           0.41712537
## CWalks         .
## LeagueN       20.28615023
## DivisionW   -116.16755870
## PutOuts        0.23752385
## Assists        .
## Errors        -0.85629148
## NewLeagueN     .
```