Chapter 4: Overfitting and Its Avoidance

- Cross validation
- C_p criterion, AIC, BIC
- Ridge Regression
- LASSO

Overfitting and Its Avoidance - Cross Validation

Let us recall k-fold cross validation first. This approach involves randomly k-fold CV dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining $k-1$ folds. The mean squared error, MSE_1, is then computed on the observations in the held-out fold. This process results in k estimates of the test error, $MSE_1, MSE_2, \dots, MSE_k$. The k-fold CV estimate is computed by averaging these values,

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i.$$

Cross-validation can also be a very useful approach in the classification setting when Y is qualitative. The k-fold CV estimate is computed by averaging error rate,

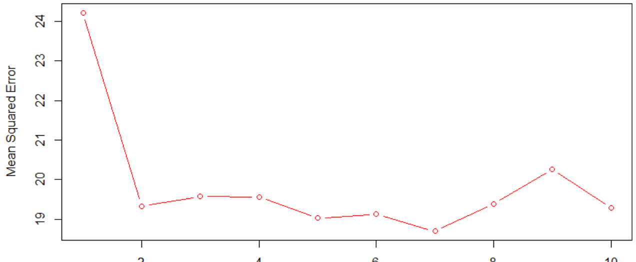
$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} Err_i,$$

where Err_i is then computed on the observations in the held-out fold.

Overfitting and Its Avoidance - Cross Validation

The cv.glm() function can also be used to implement *k*-fold CV. It is part of the boot library. Below we use k = 10, a common choice for k, on the Auto data set. We once again set a random seed and initialize a vector in which we will store the CV errors corresponding to the polynomial fits of orders one to ten.

```
    > set.seed(17)
    > cvError = rep (0, 10)
    > for (i in 1:10) {
    +   glmFit = glm(mpg ~ poly(horsepower ,i), data = Auto)
    +   cvError[i] = cv.glm(Auto, glmFit, K = 10)$delta[1]}
    > plot(1:10, cvError, type = "b", xlab = "Degree of Polynomial",
      ylab = "Mean Squared Error", col= 2)
```



Overfitting and Its Avoidance - C_p, AIC, BIC

In order to select the best model with respect to test error, we need to estimate this test error. We can indirectly estimate test error by making an adjustment to the training error to account for the bias due to overfitting. C_p, AIC and BIC can be seen as estimators of test MSE. For a fitted least squares model containing d predictors, the C_p estimate of test MSE is computed using the equation

$$C_p = \frac{1}{n}(RSS(d) + 2d\hat{\sigma^2}),$$

where $\hat{\sigma^2}$ is the estimated variance for ϵ with the full model. If the errors are Gaussian, AIC is given by

$$AIC = \log(\hat{\sigma}_d^2) + 2d/n$$

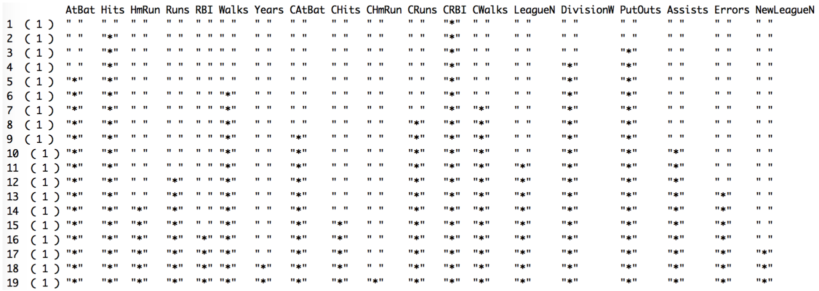
BIC is derived from a Bayesian point of view, but ends up looking similar to C_p (and AIC) as well. For the least squares model with d predictors, the BIC is given by

$$BIC = \log(\hat{\sigma}_d^2) + \log(n)d/n.$$

Overfitting and Its Avoidance - C_p, AIC, BIC

The regsubsets() function (part of the leaps library) performs best sub-set selection by identifying the best model that contains a given number of predictors, where best is quantified using RSS. The summary() command outputs the best set of variables for each model size.

```
> library(leaps)
> Hitters0 = na.omit(Hitters)
> hitSubreg = regsubsets(Salary ~ ., data = Hitters0, nvmax =
 19, method = "forward")
> regSummary = summary(hitSubreg)
```

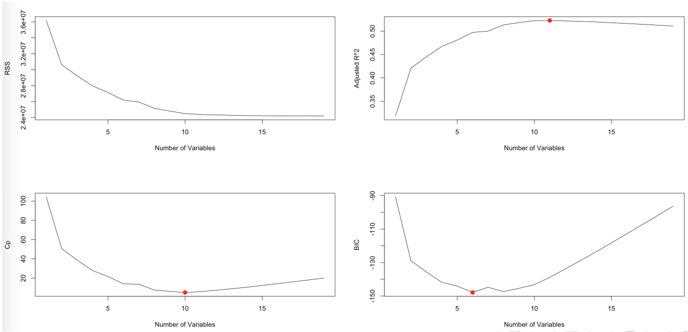


```
1         AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     " "       " "     " "     " "    " "
2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     " "       " "     " "     " "    " "
3  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     " "       "*"     " "     " "    " "
4  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     "*"       "*"     " "     " "    " "
5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     "*"       "*"     " "     " "    " "
6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"  " "    " "     "*"       "*"     " "     " "    " "
7  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"  "*"    " "     "*"       "*"     " "     " "    " "
8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    "*"   "*"  "*"    " "     "*"       "*"     " "     " "    " "
9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    " "     "*"       "*"     " "     " "    " "
10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    " "     "*"       "*"     "*"     " "    " "
11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    "*"     "*"       "*"     "*"     " "    " "
13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"   "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"   "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

Overfitting and Its Avoidance - C_p, AIC, BIC

Plotting RSS, adjusted R^2, C_p, and BIC for all of the models at once will help us decide which model to select. Note the type = "l" option tells R to connect the plotted points with lines. As expected, the RSS (resp. R^2) statistic decreases (resp. increase) monotonically as more variables are included.

```r
plot(regSummary$adjr2, xlab = "Number of Variables", ylab = "
 Adjusted R^2", type = "l")
points(which.max(regSummary$adjr2), max(regSummary$adjr2), col =
  2, cex = 2, pch = 20)
```



Overfitting and Its Avoidance - Ridge Regression

Ridge regression is very similar to least squares, except that the coefficients are estimated by minimizing a slightly different quantity. In particular, the ridge regression coefficient estimates $\hat{\beta}^R$ are the values that minimize

$$\sum_{i=1}^{n}\left(y_i - \beta_0 \sum_{j=1}^{p}\beta_j x_{ij}\right) + \lambda\sum_{j=1}^{p}\beta_j^2 = RSS + \lambda\sum_{j=1}^{p}\beta_j^2,$$

where $\lambda > 0$ is a tuning parameter, to be determined separately.

Ridge regression's advantage over least squares is rooted in the bias-variance trade-off. In general, in situations where the relationship between the response and the predictors is close to linear, the least squares estimates will have low bias but may have high variance. This means that a small change in the training data can cause a large change in the least squares coefficient estimates. Ridge regression works best in situations where the least squares estimates have high variance.

Overfitting and Its Avoidance - Ridge Regression

We will use the glmnet package in order to perform ridge regression and the lasso. The main function in this package is glmnet(), which can be used glmnet() to fit ridge regression models, lasso models, and more. In particular, we must pass in an x matrix as well as a y vector, and we do not use the $y \sim x$ syntax. Before proceeding ensure that the missing values have been removed from the data.

```
> Hitters = na.omit(Hitters)
> x = model.matrix(Salary~., Hitters)[, -1]
> y = Hitters$Salary
```

The model.matrix() function is particularly useful for creating x; not only does it produce a matrix corresponding to the 19 predictors but it also automatically transforms any qualitative variables into dummy variables.

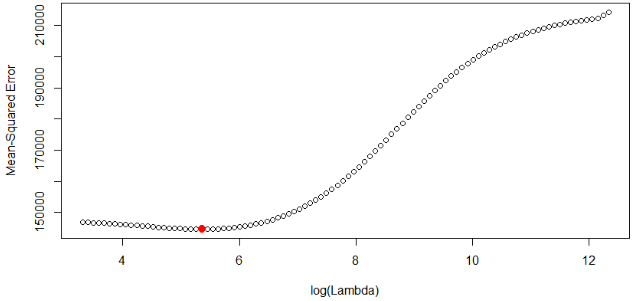
We now split the samples into a training set and a test set in order to estimate the test error of ridge regression.

```
> train = sample(1:nrow(x), nrow(x)/2)
> test = -train
```

Overfitting and Its Avoidance - Ridge Regression

It would be better to use cross-validation to choose the tuning parameter λ. We can do this using the built-in cross-validation function, cv.glmnet().

```
> cvRidgeLR = cv.glmnet(x[train, ], y[train], alpha = 0)
> bestlamda = cvRidgeLR$lambda.min
> plot(log(cvRidgeLR$lambda), cvRidgeLR$cvm, xlab = "log(Lambda)
", ylab = "Mean−Squared Error")
> points(log(cvRidgeLR$lambda[which.min(cvRidgeLR$cvm)]), min(
cvRidgeLR$cvm), col = 2, cex = 2, pch = 20)
```



Overfitting and Its Avoidance - Ridge Regression

Now we apply glmnet() function to fit the data by a ridge regression on a training set with the best lambda. The ridge regression coefficients are stored in a matrix that can be accessed by coef().

```
> ridge_fit = glmnet(x[train,], y[train], lambda = bestlambda,
  alpha = 0)
> coef(ridge_fit)[1:nrow(coef(ridge_fit)),]
 (Intercept)          AtBat          Hits           HmRun
-1.202551e+02   1.326536e-01   1.270136e+00   1.095786e+00
 Runs            RBI            Walks          Years
 1.459869e+00   1.137560e+00   2.511267e+00   1.828044e+00
 CAtBat          CHits          CHmRun         CRuns
 9.202255e-03   5.261533e-02   3.601732e-01   1.013819e-01
 CRBI           CWalks         LeagueN        DivisionW
 1.241357e-01   5.967940e-02  -7.040575e+00  -5.180590e+01
 PutOuts        Assists        Errors         NewLeagueN
 1.900937e-01  -1.121992e-01  -1.093627e+00  -3.333163e+00
```

Overfitting and Its Avoidance - Ridge Regression

Now we apply predict() function to evaluate the test MSE of ridge regression. Then we fit a linear regression model on the same training set, and evaluate its MSE on the test set. Finally we can compare both MSEs from the ridge regression and the linear regression respectively on the same test set.

```
> ridge_pred = predict(ridge_fit, newx = x[-train, ])

> trainNew = data.frame(y[train], x[train, ])
> linearRegression = lm(y.train. ~., trainNew)
> lm_pred = predict(linearRegression, as.data.frame(x[-train,]))
> c(mean((ridge_pred-y[test])^2), mean((lm_pred-y[test])^2))
[1] 138195.6 147210.8
```

At the least squares coefficient estimates, which correspond to ridge regression with $\lambda = 0$, the variance is high but there is no bias. But as λ increases, the shrinkage of the ridge coefficient estimates leads to a substantial reduction in the variance of the predictions, at the expense of a slight increase in bias. Hence, the results of test MSE illustrates that the ridge regression improve over least square since the ridge regression has advantage in the bias-variance trade-off.

Overfitting and Its Avoidance - LASSO Regression

The ridge regression will always generate a model involving all predictors. Increasing the value of λ will tend to reduce the magnitudes of the coefficients, but will not result in exclusion of any of the variables. The lasso is a relatively recent alternative to ridge regression that over comes this disadvantage. The lasso coefficients, $\hat{\beta}_{\lambda}^{L}$, minimise the quantity

$$\sum_{i=1}^{n}\left(y_i - \beta_0 \sum_{j=1}^{p}\beta_j x_{ij}\right) + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|.$$

Neither ridge regression nor the lasso will universally dominate the other. In general, one might expect the lasso to perform better in a setting where a relatively small number of predictors have substantial coefficients, and the remaining predictors have coefficients that are very small or that equal zero. Ridge regression will perform better when the response is a function of many predictors, all with coefficients of roughly equal size.

Overfitting and Its Avoidance - LASSO Regression

The test MSE of the lasso, is very similar to the test MSE of ridge regression with λ chosen by cross-validation. However, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse.

```
> set.seed(1234)
> cvLassoLR = cv.glmnet(x[train , ], y[train], alpha = 1)
> bestlambda = cvLassoLR$lambda.min
> lasso_fit = glmnet(x[train ,], y[train], lambda = bestlambda,
 alpha = 1)
> lasso_pred = predict(lasso_fit, newx = x[-train , ])
> lassoCoef = coef(lasso_fit)[1:nrow(coef(ridge_fit)),]
> lassoCoef[lassoCoef != 0]
(Intercept)         AtBat          Hits         Walks
-134.9871872    -0.8860796     5.8303572     4.3576493
CRuns           CRBI        LeagueN     DivisionW
0.1378442     0.4524613    -2.2945621   -56.6045378
PutOuts        Assists    NewLeagueN
0.2354795    -0.2191321   -11.8393033
>
> c(mean((lasso_pred-y[test])^2), mean((ridge_pred-y[test])^2),
 mean((lm_pred-y[test])^2))
[1] 139983.5 138195.6 147210.8
```