02. Linear Models for Quantitative Outcomes

Table of Contents

Linear Model

- The linear regression is defined as

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

- Linear regression assumes that the dependence of Y on X_1, X_2, $\cdots$, X_p is linear.
- True regression functions are never linear!
- Although it may seem overly simplistic, linear regression is extremely useful both conceptually and practically.

Linear Model

- Despite its simplicity, the linear model has distinct advantages in terms of its interpretability and often shows good predictive performance.
- Hence we discuss some ways in which the linear model can be improved, by replacing ordinary least squares (OLS) fitting with some alternative fitting procedures.
- The OLS minimizes the residual sum of squares (RSS). i.e.,

$$\hat{\boldsymbol{\beta}}^{OLS} = \underset{\boldsymbol{\beta} \in \mathbb{R}^{p+1}}{\arg\min}\, \mathsf{RSS}(\boldsymbol{\beta})$$

where

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

Ordinary Least Square

- Gauss-Markov Theorem: OLS estimates for $\boldsymbol{\beta}$ have the smallest variance among all linear unbiased estimates.
- Problems in multiple linear regression
  - Some coefficients in $\boldsymbol{\beta}$ are not well estimated/ have large variance for correlated predictors, since correlated variables carry the same information regarding the response.
  - OLS cannot be computed when $n < p$ (high-dimensional data).
  - OLS has relatively large variance.
- Alternative approach
  - Prediction accuracy can be improved by setting some coefficients to 0 in high-dimensional predictors.
  - For interpretation, we need to determine a smaller subset that exhibits the strongest effects on the response from a large number of predictors.

```r
set.seed(123)
n <- 100
pp <- c(10, 50, 80, 95, 97, 98, 99)
B <- matrix(0, 100, length(pp))
for (i in 1:100) {
    for (j in 1:length(pp)) {
        beta <- rep(0, pp[j])
        beta[1] <- 1
        x <- matrix(rnorm(n*pp[j]), n, pp[j])
        y <- x %*% beta + rnorm(n)
        g <- lm(y~x)
        B[i,j] <- g$coef[2]
    }
}
```

```r
boxplot(B, col="orange", boxwex=0.6, ylab="Coefficient estimates",
        names=pp, xlab="The number of predictors", ylim=c(-5,5))
abline(h=1, col=2, lty=2, lwd=2)
apply(B, 2, mean)
apply(B, 2, var)
```

Three Classes of Methods

- Subset Selection: We identify a subset of the p predictors that we believe to be related to the response. We then fit a model using OLS on the reduced set of variables.
- Shrinkage: We fit a model involving all p predictors, but the estimated coefficients are shrunken towards zero relative to the OLS estimates. This shrinkage (also known as regularization) has the effect of reducing variance and can also perform variable selection.
- Dimension Reduction: We project the p predictors into a M-dimensional subspace, where $M < p$. This is achieved by computing M different linear combinations, or projections, of the variables. Then these M projections are used as predictors to fit a linear regression model by OLS.

Deciding on the Important Variables

- The most direct approach is called all subsets or best subsets regression: we compute the least squares fit for all possible subsets and then choose between them based on some criterion that balances training error with model size.
- However we often can't examine all possible models, since they are 2^p of them; for example when $p = 40$ there are over a billion models!
- Instead we need an automated approach that searches through a subset of them: Forward selection and Backward selection.

Best Subset Selection

1. Let $\mathcal{M}_0$ denote the null model, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots, p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here best is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_p$, using cross-validated prediction error, C_p (AIC), BIC or adjusted R^2.

Example: Hitters Data

- We apply the best subset selection approach to the `Hitters` data. We wish to predict a baseball player's `Salary` on the basis of various statistics associated with performance in the previous year.
- First of all, we note that the `Salary` variable is missing for some of the players. The `is.na()` function can be used to identify the missing observations. It returns a vector of the same length as the input vector, with a `TRUE` for any elements that are missing, and a `FALSE` for non-missing elements.
- The `sum()` function can then be used to count all of the missing elements.

```
library(ISLR)
names(Hitters)
dim(Hitters)
sum(is.na(Hitters$Salary))
```

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

```
library(leaps)
fit <- regsubsets(Salary ~ ., Hitters)
summary(fit)
sg <- summary(fit)
names(sg)
dim(sg$which)
sg$which
```

```
plot(fit)
plot(fit, scale="Cp")
```

```
big <- regsubsets(Salary ~ ., data=Hitters, nvmax=19, nbest=10)
sg <- summary(big)
dim(sg$which)
sg.size <- as.numeric(rownames(sg$which))
table(sg.size)
```

```
sg.rss <- tapply(sg$rss, sg.size, min)
w1 <- which.min(sg.rss)
sg.rsq <- tapply(sg$rsq, sg.size, max)
w2 <- which.max(sg.rsq)
```

```
par(mfrow=c(1,2))
plot(1:19, sg.rss, type="b", xlab="Number of Predictors",
     ylab="Residual Sum of Squares", col=2, pch=19)
points(w1, sg.rss[w1], pch="x", col="blue", cex=2)
plot(1:19, sg.rsq, type="b", xlab="Number of Predictors",
     ylab=expression(R^2), col=2, pch=19)
points(w2, sg.rsq[w2], pch="x", col="blue", cex=2)
```

Stepwise Selection

- Forward selection
  - Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model.
  - In particular, at each step the variable that gives the greatest additional improvement to the fit is added to the model.
  - However, it is not guaranteed to find the best possible model out of all 2^p models containing subsets of the p predictors.
- Backward selection
  - Like forward stepwise selection, backward stepwise selection provides an efficient alternative to best subset selection.
  - However, unlike forward stepwise selection, it begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time.

Forward Stepwise Selection

1. Let $\mathcal{M}_0$ denote the null model, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 0, 1, \ldots, p-1$:
   (a) Consider all $p-k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   (b) Choose the best among these $p-k$ models, and call it $\mathcal{M}_{k+1}$. Here best is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_p$, using cross-validated prediction error, C_p (AIC), BIC or adjusted R^2.

Backward Stepwise Selection

1. Let $\mathcal{M}_p$ denote the full model, which contains all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   (a) Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$ for a total of $k-1$ predictors.
   (b) Choose the best among these k models, and call it $\mathcal{M}_{k-1}$. Here best is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_p$, using cross-validated prediction error, C_p (AIC), BIC or adjusted R^2.

More on Backward Stepwise Selection

- Like forward stepwise selection, the backward selection approach searches through only $1 + p(p+1)/2$ models, and so can be applied in settings where p is too large to apply best subset selection.
- Like forward stepwise selection, backward stepwise selection is not guaranteed to yield the best model containing a subset of the p predictors.
- Backward selection requires that the number of samples n is larger than the number of variables p (so that the full model can be fit). In contrast, forward stepwise can be used even when $n < p$, and so is the only viable subset method when p is very large.

```
g.full <- regsubsets(Salary ~., data=Hitters)
g.forw <- regsubsets(Salary ~., data=Hitters, method="forward")
g.back <- regsubsets(Salary ~., data=Hitters, method="backward")
```

```
full <- summary(g.full)$which[,-1]
full[full==TRUE] <- 1
forw <- summary(g.forw)$which[,-1]
forw[forw==TRUE] <- 1
back <- summary(g.back)$which[,-1]
back[back==TRUE] <- 1
```

```
full
forw
back
```

```
coef(g.full, 1:5)
coef(g.forw, 1:5)
coef(g.back, 1:5)
```

Choosing the Optimal Model

- The model containing all of the predictors will always have the smallest RSS and the largest R^2, since these quantities are related to the training error.
- We wish to choose a model with low test error, not a model with low training error. Recall that training error is usually a poor estimate of test error.
- Therefore, RSS and R^2 are not suitable for selecting the best model among a collection of models with different numbers of predictors.

Estimating Test Error: Two Approaches

- 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, BIC and adjusted R^2 adjust the training error for the model size, and can be used to select among a set of models with different numbers of variables.
- We can directly estimate the test error, using either a validation set approach or a cross-validation approach, as discussed in previous lectures.

Mallow's C_p and AIC

- Mallow's C_p statistics

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

  where d is the total number of parameters used and $\hat{\sigma}^2$ is an estimate of the variance of the error ϵ.

- Akaike Information Criterion (AIC)

$$\mathsf{AIC} = -2\log L + 2d$$

  where L is the maximized value of the likelihood function for the estimated model. The AIC criterion is defined for a large class of models fit by maximum likelihood.

- For the linear model with normal errors, maximum likelihood and least squares are the same thing, so C_p and AIC are equivalent.

BIC

- Bayesian Information Criterion (BIC) of a least squares model

$$\text{BIC} = \frac{1}{n}\left(\text{RSS} + \log(n)d\hat{\sigma}^2\right)$$

- Like C_p, the BIC will tend to take on a small value for a model with a low test error, and so generally we select the model that has the lowest value.
- Notice that BIC replaces the $2d\hat{\sigma}^2$ used by C_p with a $\log(n)d\hat{\sigma}^2$ term, where n is the number of observations.
- Since $\log(n) > 2$ for any $n > 7$, the BIC statistic generally places a heavier penalty on models with many variables, and hence results in the selection of smaller models than C_p.

Adjusted R^2

- For a least squares model with d variables, the adjusted R^2 statistic is calculated as

$$\text{Adjusted } R^2 = 1 - \frac{\text{RSS}/(n-d-1)}{\text{TSS}/(n-1)},$$

  where TSS is the total sum of squares.
- Unlike C_p, AIC, and BIC, for which a small value indicates a model with a low test error, a large value of adjusted R^2 indicates a model with a small test error.
- Unlike the R^2 statistic, the adjusted R^2 statistic pays a price for the inclusion of unnecessary variables in the model. Note that RSS$/(n-d-1)$ may increase or decrease due to the presence of d in the denominator, while RSS always decreases as the number of variables in the model increases.

```
sg.cp <- tapply(sg$cp, sg.size, min)
w3 <- which.min(sg.cp)
sg.bic <- tapply(sg$bic, sg.size, min)
w4 <- which.min(sg.bic)
sg.adjr2 <- tapply(sg$adjr2, sg.size, max)
w5 <- which.max(sg.adjr2)
```

```
par(mfrow=c(1,3))
plot(1:19, sg.cp, type="b", xlab ="Number of Predictors",
     ylab=expression(C[p]), col=2, pch=19)
points(w3, sg.cp[w3], pch="x", col="blue", cex=2)
plot(1:19, sg.bic, type="b", xlab ="Number of Predictors",
     ylab="Bayesian information criterion", col=2, pch=19)
points(w4, sg.bic[w4], pch="x", col="blue", cex=2)
plot(1:19, sg.adjr2, type="b", xlab ="Number of Predictors",
     ylab=expression(paste("Adjusted ", R^2)), col=2, pch=19)
points(w5, sg.adjr2[w5], pch="x", col="blue", cex=2)
```

```
model1 <- coef(big, which.min(sg$rss))
model2 <- coef(big, which.max(sg$rsq))
model3 <- coef(big, which.max(sg$adjr2))
model4 <- coef(big, which.min(sg$cp))
model5 <- coef(big, which.min(sg$bic))
```

```
RES <- matrix(0, 20, 5)
rownames(RES) <- names(model1)
colnames(RES) <- c("rss", "rsq", "adjr2", "cp", "bic")
```

```
for (i in 1:5) {
    model <- get(paste("model", i, sep=""))
    w <- match(names(model), rownames(RES))
    RES[w, i] <- model
}
RES
```

```
apply(RES, 2, function(t) sum(t!=0)-1)
```

Validation and Cross-Validation

- Each of the procedures returns a sequence of models $\mathcal{M}_k$ indexed by model size $k = 0, 1, 2, \ldots$. Our job here is to select $\hat{k}$. Once selected, we will return model $\mathcal{M}_{\hat{k}}$.
- We compute the validation set error or the cross-validation error for each model $\mathcal{M}_k$ under consideration, and then select the k for which the resulting estimated test error is smallest.
- This procedure has an advantage relative to AIC, BIC, C_p, and adjusted R^2, in that it provides a direct estimate of the test error, and doesn't require an estimate of the error variance σ^2.
- It can also be used in a wider range of model selection tasks, even in cases where it is hard to pinpoint the model degrees of freedom (e.g. the number of predictors in the model) or hard to estimate the error variance σ^2.

Validation and Cross-Validation

- The validation errors were calculated by randomly selecting the observations as the training set, and the remainder as the validation set.
- The cross-validation errors were computed using $k = 10$ folds.
- Note that the validation method result in a seven-variable model, while the cross-validation method result in a ten-variable model when both start with `set.seed(1)`.
- In the cross-validation, we can select a model using the one-standard-error rule. We first calculate the standard error of the estimated test MSE for each model size, and then select the smallest model for which the estimated test error is within one standard error of the lowest point on the curve.

```
set.seed(1)
train <- sample(c(TRUE, FALSE), nrow(Hitters), replace=TRUE)
test <- (!train)
g1 <- regsubsets(Salary ~ ., data=Hitters[train, ], nvmax=19)
test.mat <- model.matrix(Salary~., data=Hitters[test, ])
val.errors <- rep(NA, 19)
for (i in 1:19) {
    coefi <- coef(g1, id=i)
    pred <- test.mat[, names(coefi)] %*% coefi
    val.errors[i] <- sqrt(mean((Hitters$Salary[test]-pred)^2))
}
val.errors
w <- which.min(val.errors)
```

```
par(mfrow=c(1,2))
plot(1:19, val.errors, type="l", col="red",
     xlab="Number of Predictors", ylab="Validation Set Error")
points(1:19, val.errors, pch=19, col="blue")
points(w, val.errors[w], pch="x", col="blue", cex=2)
```

```r
set.seed(1234)
N <- 8
```

```r
ERR <- matrix(0, 19, N)
for (k in 1:N) {
    tr <- sample(c(TRUE, FALSE), nrow(Hitters), replace=TRUE)
    tt <- (!tr)
    g <- regsubsets(Salary ~ ., data=Hitters[tr, ], nvmax=19)
    tt.mat <- model.matrix(Salary~., data=Hitters[tt, ])
    for (i in 1:19) {
        coefi <- coef(g, id=i)
        pred <- tt.mat[, names(coefi)] %*% coefi
        ERR[i,k] <- sqrt(mean((Hitters$Salary[tt]-pred)^2))
    }
}
```

```r
matplot(ERR, type="l", col="red", xlab="Number of Predictors",
        lty=1, ylab="Validation Set Error")
apply(ERR, 2, which.min)
```

```
## Define new "predict" function on regsubset
predict.regsubsets <- function(object, newdata, id, ...) {
    form <- as.formula(object$call[[2]])
    mat <- model.matrix(form, newdata)
    coefi <- coef(object, id=id)
    xvars <- names(coefi)
    mat[, xvars] %*% coefi
}
```

```
set.seed(1)
K <- 10
n <- nrow(Hitters)
fd <- sample(rep(1:K, length=n))
cv.errors <- matrix(NA , n, 19, dimnames=list(NULL, paste(1:19)))
for (i in 1:K) {
    fit <- regsubsets(Salary~., Hitters[fd!=i, ], nvmax=19)
    for (j in 1:19) {
        pred <- predict(fit, Hitters[fd==i, ], id=j)
        cv.errors[fd==i, j] <- (Hitters$Salary[fd==i]-pred)^2
    }
}
```

```
sqrt(apply(cv.errors, 2, mean))
K.ERR <- sqrt(apply(cv.errors, 2, mean))
ww <- which.min(K.ERR)
```

```
par(mfrow=c(1,2))
plot(1:19, K.ERR, type="l", col="red",
     xlab="Number of Predictors", ylab="Cross-Validation Error")
points(1:19, K.ERR, pch=19, col="blue")
points(ww, K.ERR[ww], pch="x", col="blue", cex=2)
```

```
## 10-fold CV with 8 different splits
N <- 8
n <- nrow(Hitters)
ERR <- matrix(0, 19, N)
```

```
set.seed(1234)
```

```
for (k in 1:N) {
    fd <- sample(rep(1:K, length=n))
    CVR <- matrix(NA , n, 19)
    for (i in 1:K) {
        f <- regsubsets(Salary~., data=Hitters[fd!=i, ], nvmax=19)
        for (j in 1:19) {
            pred <- predict(f, Hitters[fd==i, ], id=j)
            CVR[fd==i, j] <- (Hitters$Salary[fd==i]-pred)^2
        }
    }
    ERR[,k] <- sqrt(apply(CVR, 2, mean))
}
```

```
matplot(ERR, type="l", col="red", xlab="Number of Predictors",
        lty=1, ylab="Cross-Validation Error")
apply(ERR, 2, which.min)
```

```
set.seed(111)
fd <- sample(rep(1:K, length=n))
CVR.1se <- matrix(NA, n, 19)
for (i in 1:K) {
    fit <- regsubsets(Salary~., Hitters[fd!=i, ], nvmax=19)
    for (j in 1:19) {
        pred <- predict(fit, Hitters[fd==i, ], id=j)
        CVR.1se[fd==i, j] <- Hitters$Salary[fd==i]-pred
    }
}
avg <- sqrt(apply(CVR.1se^2, 2, mean))
se <- apply(CVR.1se, 2, sd)/sqrt(n)
PE <- cbind(avg - se, avg, avg + se)
```

```
data.frame(lwr=PE[,1], mean=PE[,2], upp=PE[,3])
which.min(PE[,2])
w <- which.min(PE[,2])
which(PE[w, 1] < PE[,2] & PE[w, 3] > PE[,2])
min(which(PE[w, 1] < PE[,2] & PE[w, 3] > PE[,2]))
```

```r
dev.off()
matplot(1:19, PE, type="b", col=c(1,2,1), lty=c(3,1,3), pch=20,
      xlab="Number of Predictors", ylab="Cross-Validation Error")
abline(h=PE[w, 1], lty=3, col="gray")
abline(h=PE[w, 3], lty=3, col="gray")
points(which.min(avg), PE[which.min(avg),2],
       pch="o",col="blue",cex=2)
up <- which(PE[,2] < PE[which.min(PE[,2]),3])
points(min(up), PE[min(up),2], pch="x", col="blue", cex=2)
```

Shrinkage Methods

- The subset selection methods use least squares to fit a linear model that contains a subset of the predictors.
- As an alternative, we can fit a model containing all p predictors using a technique that constrains or regularizes the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero.
- It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance.
- Example :
  - Ridge regression
  - Lasso (Least absolute shrinkage and selection operator).

Ridge Regression

- Recall that the OLS fitting procedure estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ using the values that minimize

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

- In contrast, the ridge regression coefficient estimates $\hat{\beta}^{ridge}$ are the values that minimize

$$\sum_{i}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \|\boldsymbol{\beta}\|_2^2 = \mathsf{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2,$$

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

Ridge Regression

- As with least squares, ridge regression seeks coefficient estimates that fit the data well, by making the RSS small.
- However, the second term, $\lambda \sum_j \beta_j^2$, called a shrinkage penalty, is small when $\beta_1, \beta_2, \cdots, \beta_p$ are close to zero, and so it has the effect of shrinking the estimates of β_j towards zero.
- The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates.
- For a grid of λ values such as

$$\lambda_{\text{max}} = \lambda_1 > \lambda_2 > \ldots > \lambda_{m-1} > \lambda_m = \lambda_{\text{min}},$$

the l_2-norm of $\hat{\boldsymbol{\beta}}$ is

$$\|\hat{\boldsymbol{\beta}}_{\lambda_1}\|_2 \leq \|\hat{\boldsymbol{\beta}}_{\lambda_2}\|_2 \leq \ldots \leq \|\hat{\boldsymbol{\beta}}_{\lambda_{m-1}}\|_2 \leq \|\hat{\boldsymbol{\beta}}_{\lambda_m}\|_2$$

Scaling of Predictors

- The standard least squares coefficient estimates are scale equivariant: multiplying X_j by a constant c simply leads to a scaling of the least squares coefficient estimates by a factor of $1/c$. In other words, regardless of how the jth predictor is scaled, $X_j\hat{\beta}_j$ will remain the same.
- In contrast, the ridge regression coefficient estimates can change substantially when multiplying a given predictor by a constant, due to the sum of squared coefficients term in the penalty part of the ridge regression objective function.
- Therefore, it is best to apply ridge regression after standardizing the predictors, using the formula

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{n^{-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

```
library(glmnet)
```

```
x0 <- model.matrix(Salary~., Hitters)[, -1]
y <- Hitters$Salary
grid <- 10^seq(10, -2, length=100)
```

```
g1 <- glmnet(x0, y, alpha=0, lambda=grid)
par(mfrow=c(1,2))
plot(g1, "lambda", label=TRUE)
```

```
fun <- function(t) sqrt(var(t)*(length(t)-1)/length(t))
sdx <- matrix(apply(x0, 2, fun), dim(x0)[2], dim(x0)[1])
x <- x0/t(sdx)
```

```
g2 <- glmnet(x, y, alpha=0, lambda=grid)
plot(g2, "lambda", label=TRUE)
data.frame(sd_g1=apply(x0, 2, sd), sd_g2=apply(x, 2, sd))
```

```
names(g2)
data.frame(lambda=g2$lambda, df=g2$df)
data.frame(log.lambda=round(log(g2$lambda), 4), df=g2$df)
dim(coef(g2))
coef(g2)[, c("s0", "s10", "s20", "s30")]
```

```
g2$lambda[50]
coef(g2)[,50]
sqrt(sum(coef(g2)[-1, 50]^2))
```

```
g2$lambda[60]
coef(g2)[,60]
sqrt(sum(coef(g2)[-1, 60]^2))
```

```
l2norm <- function(t) sqrt(sum(t^2))
l2 <- apply(coef(g2)[-1,], 2, l2norm)
data.frame(log_lambda=round(log(g2$lambda), 4),
           l2norm=round(l2, 4))
```

Lasso

- Ridge regression does have one obvious disadvantage: unlike subset selection, which will generally select models that involve just a subset of the variables, ridge regression will include all p predictors in the final model.
- The Lasso is a relatively recent alternative to ridge regression that overcomes this disadvantage. The lasso coefficients, $\hat{\beta}^{lasso}$, minimize the quantity

$$\sum_{i}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\|\boldsymbol{\beta}\|_1 = \mathsf{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|,$$

- As with ridge regression, the lasso shrinks the coefficient estimates towards zero.

Lasso

- In lasso, the l_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large.
- For a grid of λ values such as

$$\lambda_{\text{max}} = \lambda_1 > \lambda_2 > \ldots > \lambda_{m-1} > \lambda_m = \lambda_{\text{min}},$$

  the number of nonzero regression coefficients (degrees of freedom: df) is

$$df(\hat{\boldsymbol{\beta}}_{\lambda_1}) = 0 \leq df(\hat{\boldsymbol{\beta}}_{\lambda_2}) \leq \ldots \leq df(\hat{\boldsymbol{\beta}}_{\lambda_{m-1}}) \leq df(\hat{\boldsymbol{\beta}}_{\lambda_m})$$

- Hence, much like best subset selection, the lasso performs variable selection.
- In lasso regression, selecting the optimal value of λ is crucial.

```
g3 <- glmnet(x, y, alpha=1)
par(mfrow=c(1,2))
plot(g3, "lambda", label=TRUE)
plot(g3, "norm", label=TRUE)
```

```
dim(coef(g3))
coef(g3)[, c("s0", "s10", "s40", "s60")]
data.frame(lambda=g3$lambda, df=g3$df)
```

```
dim(g3$beta)
df2 <- apply(g3$beta, 2, function(t) sum(t!=0))
data.frame(df1=g3$df, df2=df2)
```

```
l1norm <- function(t) sum(abs(t))
l1 <- apply(g3$beta, 2, l1norm )
data.frame(log_lambda=round(log(g3$lambda), 4),
           l1norm=round(l1, 4))
```

The Variable Selection Property of Lasso

- Why is it that lasso, unlike ridge regression, results in coefficient estimates that are exactly equal to zero?
- One can show that the lasso and ridge regression coefficient estimates solve the problems

$$\underset{\beta}{\text{minimize}} \sum_{i}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq s$$

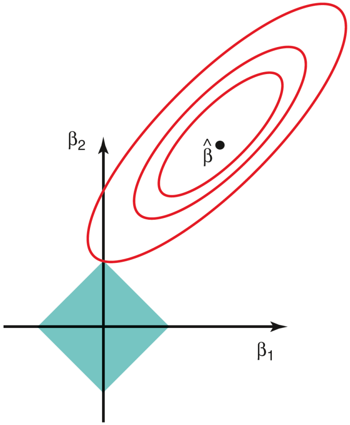
and

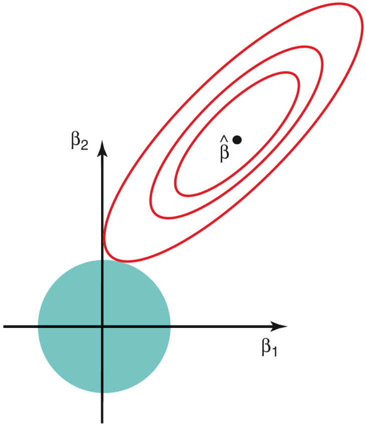
$$\underset{\beta}{\text{minimize}} \sum_{i}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq s,$$

respectively.

Lasso and Ridge Picture

- When $p = 2$, Lasso and ridge regression are





Selecting the Tuning Parameter

- As for subset selection, for ridge regression and lasso we require a method to determine which of the models under consideration is best.
- That is, we require a method selecting a value for the tuning parameter λ or equivalently, the value of the constraint s.
- Cross-validation provides a simple way to tackle this problem. We choose a grid of λ values, and compute the cross-validation error rate for each value of λ.
- We then select the tuning parameter value for which the cross-validation error is smallest.
- For a smaller (sparser) model, one-standard-error rule can be applied.

Training Set

- Training set is used to train the model, i.e., estimate $p+1$ regression coefficients. Suppose that we have m different models
$$\hat{f}_{\lambda_1}(x), \hat{f}_{\lambda_2}(x), \hat{f}_{\lambda_3}(x), \ldots, \hat{f}_{\lambda_m}(x)$$
- The l-th model has $\hat{\boldsymbol{\beta}}_l = \{\hat{\beta}_{0,l}, \hat{\beta}_{1,l}, \ldots, \hat{\beta}_{p,l}\}$ such that
$$\hat{f}_{\lambda_l}(x) = \hat{\beta}_{0,l} + \hat{\beta}_{1,l}x_1 + \cdots + \hat{\beta}_{p,l}x_p$$
- In general, for $\lambda_1 = \lambda_{\max}$, i.e., $l = 1$,
$$\hat{\beta}_{1,1} = \hat{\beta}_{2,1} = \cdots = \hat{\beta}_{p,1} = 0.$$
So, $\hat{f}_{\lambda_1}(x) = \hat{\beta}_{0,l}$.
- $\hat{\boldsymbol{\beta}}_l$ should be estimated based only on training set for each l.

Validation Set

- Validation set is used to assess the model performance. The validation set should not used in the model building process.
- Given the validation data $T = \{x_i, y_i\}$ for $i \in \{1, \ldots, N\}$, the mean squared error (MSE) of λ_l is

$$MSE_{\lambda_l} = \frac{1}{N} \sum_{i \in T} \left(y_i - \hat{f}_{\lambda_l}(x_i) \right)^2$$

- We can find the best model among m models as comparing the validation set error of m models.

$$\hat{\lambda} = \arg\min_{\lambda_1, \ldots, \lambda_m} MSE_{\lambda_l}$$

- The final model is then $\hat{f}_{\hat{\lambda}}(x)$ with full data set.

K-fold Cross-validation

- Suppose that y_i is a quantitative value of the i-th individual.
- The K-fold CV procedure:
  1. Randomly separate n samples into K folds: $C_1, C_2, \ldots, C_K$
  2. For each λ, compute regression coefficients based on C_{-k},

     $$\hat{\boldsymbol{\beta}}_{\lambda_1}^{[-k]}, \hat{\boldsymbol{\beta}}_{\lambda_2}^{[-k]}, \ldots, \hat{\boldsymbol{\beta}}_{\lambda_{m-1}}^{[-k]}, \hat{\boldsymbol{\beta}}_{\lambda_m}^{[-k]}$$

     Note that C_{-k} is an observation set with part k removed.
  3. Compute cross-validation error (CVE) for each λ.

     $$\mathsf{CVE}(\lambda_l) = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in C_k} \left(y_i - x_i^{\mathrm{T}} \hat{\boldsymbol{\beta}}_{\lambda_l}^{[-k]} \right)^2$$

     for $l = 1, 2, \ldots, m$.
  4. Pick up the optimal $\hat{\lambda}_l$ that minimizes $\mathsf{CVE}(\hat{\lambda}_l)$.

One-standard-error Rule in Regularization

- Choose a smaller model: one-standard-error rule
- Calculate the standard error of the estimated test MSE for each λ.

$$sd\left(\mathsf{CVE}(\lambda_l)\right) = \sqrt{\frac{1}{n-1}\sum_{k=1}^{K}\left(\mathsf{MSE}_k(\lambda_l) - \frac{1}{n}\sum_{k=1}^{K}\mathsf{MSE}_k(\lambda_l)\right)^2}$$

where

$$\mathsf{MSE}_k(\lambda_l) = \sum_{i \in C_k}\left(y_i - x_i^{\mathrm{T}}\hat{\boldsymbol{\beta}}_{\lambda_l}^{[-k]}\right)^2$$

- Select the smallest model (largest λ) for which the test error is within one standard error of the lowest point on the curve.

$$[\min(\mathsf{CVE}) - sd\left(\mathsf{CVE}\right), \min(\mathsf{CVE}) + sd\left(\mathsf{CVE}\right)]$$

```
set.seed(123)
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
y.test <- y[test]
```

```
grid <- 10^seq(10, -2, length=100)
r1 <- glmnet(x[train, ], y[train], alpha=0, lambda=grid)
ss <- 0:(length(r1$lambda)-1)
Err <- NULL
```

```
for (i in 1:length(r1$lambda)) {
    r1.pred <- predict(r1, s=ss[i], newx=x[test, ])
    Err[i] <- mean((r1.pred - y.test)^2)
}
wh <- which.min(Err)
lam.opt <- r1$lambda[wh]
```

```
r.full <- glmnet(x, y, alpha=0, lambda=grid)
r.full$beta[,wh]
predict(r.full, type="coefficients", s=lam.opt)
```

```
set.seed(1)
cv.r <- cv.glmnet(x, y, alpha=0, nfolds=10)
names(cv.r)
```

```
cbind(cv.r$cvlo, cv.r$cvm, cv.r$cvup)
dev.off()
plot(cv.r)
```

```
log(cv.r$lambda.min)
log(cv.r$lambda.1se)
```

```
which(cv.r$lambda==cv.r$lambda.min)
which(cv.r$lambda==cv.r$lambda.1se)
```

```
b.min <- predict(cv.r, type="coefficients", s=cv.r$lambda.min)
b.1se <- predict(cv.r, type="coefficients", s=cv.r$lambda.1se)
cbind(b.min, b.1se)
c(sqrt(sum(b.min[-1]^2)), sqrt(sum(b.1se[-1]^2)))
```

```
set.seed(2)
cv.l <- cv.glmnet(x, y, alpha=1, nfolds=10)
plot(cv.l)
```

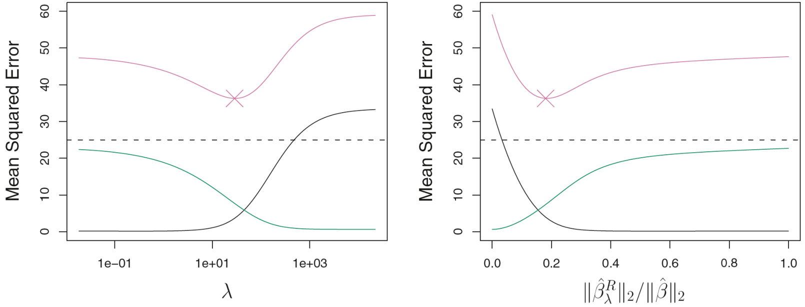
```
log(cv.l$lambda.min)
log(cv.l$lambda.1se)
```

```
which(cv.l$lambda==cv.l$lambda.min)
which(cv.l$lambda==cv.l$lambda.1se)
```

```
b.min <- predict(cv.l, type="coefficients", s=cv.l$lambda.min)
b.1se <- predict(cv.l, type="coefficients", s=cv.l$lambda.1se)
cbind(b.min, b.1se)
```

```
c(sum(abs(b.min[-1])), sum(abs(b.1se[-1])))
```

Ridge: The Bias-Variance tradeoff



- Simulated data with $n = 50$ observations, $p = 45$ predictors, all having nonzero coefficients.
- Squared bias (black), variance (green), and test MSE (purple) for the ridge regression predictions on a simulated data set.
- The purple crosses indicate the ridge regression models for which the MSE is smallest.

```
set.seed(1234)
K <- 100
p <- 40
n <- 50
beta <- runif(p, -1, 1)
lam <- 10^seq(3, -3, length.out=50)
x <- matrix(rnorm(n * p), n, p)
```

```
bhat <- array(0, c(p, length(lam), K))
for (i in 1:K) {
    y <- x %*% beta + rnorm(n)
    fit <- glmnet(x, y, alpha=0, lambda=lam)
    bhat[,,i] <- as.matrix(fit$beta)
}
```

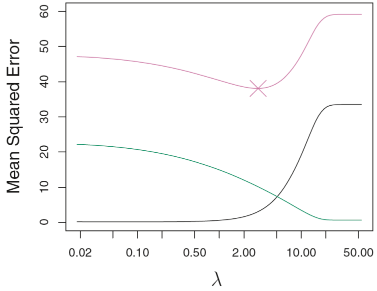
```
MSE0 <- Bias0 <- Vars0 <- matrix(0, p, length(lam))
for (k in 1:length(lam)) {
    MS <- (bhat[,k,] - matrix(beta, p, K))^2
    MSE0[,k] <- apply(MS, 1, mean)
    Bias0[,k] <- abs(apply(bhat[,k,], 1, mean) - beta)
    Vars0[,k] <- apply(bhat[,k,], 1, var)
}
```

```
MSE <- apply(MSE0, 2, mean)
Bias <- apply(Bias0, 2, mean)
Vars <- apply(Vars0, 2, mean)
```

```
MAT <- cbind(Bias^2, Vars, MSE)
data.frame(lambda=round(lam, 3), MSE=MAT[,3])
```

```
matplot(MAT, type="l", col=c(3,4,2), lty=1, xaxt="n",
        xlab=expression(lambda), ylab="Mean Squared Error")
legend("topright", c("Bias", "Variance", "MSE"), col=c(3,4,2),
       lty=1)
w <- which.min(MAT[,3])
abline(v=w, col="gray", lty=2)
points(w, MAT[w, 3], pch=4, col="red", cex=2)
cc <- seq(1, 50, 8)
axis(1, at=cc, labels=round(lam[cc],4))
```

Lasso: The Bias-Variance tradeoff



- Simulated data with $n = 50$ observations, $p = 45$ predictors, only 10 having nonzero coefficients.
- Squared bias (black), variance (green), and test MSE (purple) for the lasso regression predictions on a simulated data set.
- The purple crosses indicate the lasso regression models for which the MSE is smallest.

```
set.seed(1234)
K <- 100
p <- 40
n <- 50
beta <- runif(p, -1, 1)
lam <- 10^seq(0.7, -4, length.out=50)
x <- matrix(rnorm(n * p), n, p)
```

```
bhat <- array(0, c(p, length(lam), K))
for (i in 1:K) {
    y <- x %*% beta + rnorm(n)
    fit <- glmnet(x, y, alpha=1, lambda=lam)
    bhat[,,i] <- as.matrix(fit$beta)
}
```

```
MSE0 <- Bias0 <- Vars0 <- matrix(0, p, length(lam))
for (k in 1:length(lam)) {
    MS <- (bhat[,k,] - matrix(beta, p, K))^2
    MSE0[,k] <- apply(MS, 1, mean)
    Bias0[,k] <- abs(apply(bhat[,k,], 1, mean) - beta)
    Vars0[,k] <- apply(bhat[,k,], 1, var)
}
```

```r
MSE <- apply(MSE0, 2, mean)
Bias <- apply(Bias0, 2, mean)
Vars <- apply(Vars0, 2, mean)
```

```r
MAT <- cbind(Bias^2, Vars, MSE)
data.frame(lambda=round(lam, 3), MSE=MAT[,3])
```

```r
matplot(MAT, type="l", col=c(3,4,2), lty=1, xaxt="n",
        xlab=expression(lambda), ylab="Mean Squared Error")
legend("topright", c("Bias", "Variance", "MSE"), col=c(3,4,2),
       lty=1)
w <- which.min(MAT[,3])
abline(v=w, col="gray", lty=2)
points(w, MAT[w, 3], pch=4, col="red", cex=2)
cc <- seq(1, 50, 8)
axis(1, at=cc, labels=round(lam[cc],4))
```

Comparison between Lasso and Ridge Regression

- Neither ridge regression nor the lasso will universally dominate the other.
  - If nonzero coefficients are large, ridge is better than lasso.
  - If nonzero coefficients are small, lasso is better than ridge.
- However, the number of predictors that is related to the response is never known a priori for real data sets.
- In general, one might expect the lasso to perform better for analysis of high-dimensional data where sparse model is generally assumed.
- Cross-validation can be used to determine which approach is better on a particular data set.

```
set.seed(123)
K <- 100; p <- 45; n <- 50
beta <- runif(p, -1, 1)
lam <- 10^seq(2, -3, -0.1)
```

```
RES1 <- RES2 <- array(0, c(n, length(lam), K))
x <- matrix(rnorm(n * p), n, p)
ty <- matrix(0, n, K)
for (i in 1:K) {
    y <- x %*% beta + rnorm(n)
    ty[,i] <- y
    gr <- sample(rep(seq(5), length=length(y)))
    yhat1 <- yhat2 <- array(0, c(n, length(lam)))
    for (j in 1:5) {
        tran <- gr!=j
        g1 <- glmnet(x[tran,], y[tran,], alpha=0, lambda=lam)
        g2 <- glmnet(x[tran,], y[tran,], alpha=1, lambda=lam)
        yhat1[!tran, ] <- predict(g1, x)[!tran,]
        yhat2[!tran, ] <- predict(g2, x)[!tran,]
    }
    RES1[,,i] <- yhat1
    RES2[,,i] <- yhat2
}
```

```
title <- c("Ridge", "Lasso")
par(mfrow=c(1,2))
for (l in 1:2) {
    RES <- get(paste("RES", l, sep=""))
    MSE0 <- Bias0 <- Vars0 <- matrix(0, n, length(lam))
    for (k in 1:length(lam)) {
        PE <- (RES[,k,] - ty)^2
        MSE0[,k] <- apply(PE, 1, mean)
        Bias0[,k] <- abs(apply(RES[,k,]-ty, 1, mean))
        Vars0[,k] <- apply(RES[,k,], 1, var)
    }
    MSE <- apply(MSE0, 2, mean)
    Bias <- apply(Bias0, 2, mean)
    Vars <- apply(Vars0, 2, mean)
    MAT <- apply(cbind(Bias^2, Vars, MSE), 2, rev)
    matplot(MAT, type="l", col=c(3,4,2), lty=1, xaxt="n",
            xlab=expression(lambda), ylab="Prediction Error",
            main=title[l])
    abline(h=min(MSE), col="gray", lty=2)
    legend("topleft", c("Bias", "Variance", "MSE"), lty=1,
           col=c(3,4,2))
    cc <- c(1, seq(11, 61, 10))
    axis(1, at=cc, labels=round(rev(lam)[cc],4))
}
```

```
## Only first 10 predictors have non-zero coefficients
set.seed(12345)
beta <- rep(0, p)
beta[1:10] <- runif(10, -1, 1)
```

```
RES1 <- RES2 <- array(0, c(n, length(lam), K))
x <- matrix(rnorm(n * p), n, p)
ty <- matrix(0, n, K)
for (i in 1:K) {
    y <- x %*% beta + rnorm(n)
    ty[,i] <- y
    gr <- sample(rep(seq(5), length=length(y)))
    yhat1 <- yhat2 <- array(0, c(n, length(lam)))
    for (j in 1:5) {
        tran <- gr!=j
        g1 <- glmnet(x[tran,], y[tran,], alpha=0, lambda=lam)
        g2 <- glmnet(x[tran,], y[tran,], alpha=1, lambda=lam)
        yhat1[!tran, ] <- predict(g1, x)[!tran,]
        yhat2[!tran, ] <- predict(g2, x)[!tran,]
    }
    RES1[,,i] <- yhat1
    RES2[,,i] <- yhat2
}
```

```
for (l in 1:2) {
    RES <- get(paste("RES", l, sep=""))
    MSE0 <- Bias0 <- Vars0 <- matrix(0, n, length(lam))
    for (k in 1:length(lam)) {
        PE <- (RES[,k,] - ty)^2
        MSE0[,k] <- apply(PE, 1, mean)
        Bias0[,k] <- abs(apply(RES[,k,]-ty, 1, mean))
        Vars0[,k] <- apply(RES[,k,], 1, var)
    }
    MSE <- apply(MSE0, 2, mean)
    Bias <- apply(Bias0, 2, mean)
    Vars <- apply(Vars0, 2, mean)
    MAT <- apply(cbind(Bias^2, Vars, MSE), 2, rev)
    matplot(MAT, type="l", col=c(3,4,2), lty=1, xaxt="n",
            xlab=expression(lambda), ylab="Prediction Error",
            main=title[l])
    abline(h=min(MSE), col="gray", lty=2)
    legend("topright", c("Bias", "Variance", "MSE"), lty=1,
           col=c(3,4,2))
    cc <- c(1, seq(11, 61, 10))
    axis(1, at=cc, labels=round(rev(lam)[cc],4))
}
```

```
PE.fun <- function(n, p, K, beta, lam, xtest, ytest) {
    yhat0 <- yhat1 <- array(0, c(n, length(lam), K))
    for (i in 1:K) {
        x <- matrix(rnorm(n * p), n, p)
        y <- x %*% beta + rnorm(n)
        g0 <- glmnet(x, y, alpha=0, lambda=lam)
        g1 <- glmnet(x, y, alpha=1, lambda=lam)
        yhat0[1:n, 1:length(lam), i] <- predict(g0, x.test)
        yhat1[1:n, 1:length(lam), i] <- predict(g1, x.test)
    }
    PE0 <- PE1 <- array(0, c(n, length(lam)))
    for (j in 1:length(lam)) {
        MSE0 <-(yhat0[,j,] - matrix(ytest, n, K))^2
        MSE1 <-(yhat1[,j,] - matrix(ytest, n, K))^2
        PE0[ ,j] <- apply(MSE0, 1, mean)
        PE1[ ,j] <- apply(MSE1, 1, mean)
    }
    Ridge <- apply(PE0, 2, mean)
    Lasso <- apply(PE1, 2, mean)
cbind(Ridge, Lasso)
}
```

```
set.seed(123)
K <- 50
p <- 200
n <- 100
lam <- 10^seq(1, -3, -0.05)
x.test <- matrix(rnorm(n * p), n, p)
```

```
beta1 <- beta2 <- runif(p, -1, 1)
ytest1 <- x.test %*% beta1 + rnorm(n)
g1 <- PE.fun(n, p, K, beta1, lam, xtest, ytest1)
apply(g1, 2, summary)
```

```
beta2[11:p] <- 0
ytest2 <- x.test %*% beta2 + rnorm(n)
g2 <- PE.fun(n, p, K, beta2, lam, xtest, ytest2)
apply(g2, 2, summary)
```

```
set.seed(111)
K <- 20
p <- 1000
n <- 100
lam <- 10^seq(2, -3, -0.05)
x.test <- matrix(rnorm(n * p), n, p)
```

```
beta1 <- beta2 <- runif(p, -1, 1)
ytest1 <- x.test %*% beta1 + rnorm(n)
g1 <- PE.fun(n, p, K, beta1, lam, xtest, ytest1)
apply(g1, 2, summary)
```

```
beta2[11:p] <- 0
ytest2 <- x.test %*% beta2 + rnorm(n)
g2 <- PE.fun(n, p, K, beta2, lam, xtest, ytest2)
apply(g2, 2, summary)
```

Introduction to Survival Analysis

- Sometimes, we observe survival times of individuals as our response variable. Although it seems that survival times are quantitative, survival analysis is completely different.
- In survival analysis, the outcome (response) variable is "time until an event occurs".
- In a five-year medical study, where patients have been treated for cancer
  - We want to predict patient survival time based on health measurements or type of treatment.
  - Some patients have survived until the end of the study: such a patient's survival time is said to be censored.
- Survival time of censored patients is at least five years, but we do not know its true value.
- Survival analysis is how to handle with censored data.

Introduction to Survival Analysis

- Although survival analysis evokes a medical study, the applications of survival analysis extend far beyond medicine.
- For example,
  - A company collect data on customers over some time period, in order to model each customer's time to cancellation, but not all customers will have canceled their subscription by the end of this time period.
  - We wish to model a person's weight for a large number of people, but the scale used to weigh those people is unable to report weights above a certain number. So, any weights that exceed that number are censored.
- Survival analysis is a very well-studied topic within statistics, due to its critical importance in a variety of applications, both in and out of medicine.

Survival and Censoring Times

- In our survival data, we have
  - T: a true survival time
  - C: a true censoring time
- The survival time is also known as failure time or event time.
- The survival time represents the time at which the event of interest occurs
  - The time at which the patient dies
  - The time at which the customer cancels his/her subscription
- The censoring time is the time at which censoring occurs.
  - The time at which the patient drops out of the study
  - The time at which the study ends

Survival and Censoring Times

- We observe either T or C for each individual. Specifically,

$$Y = \min(T, C)$$

- We also observe a status indicator δ,

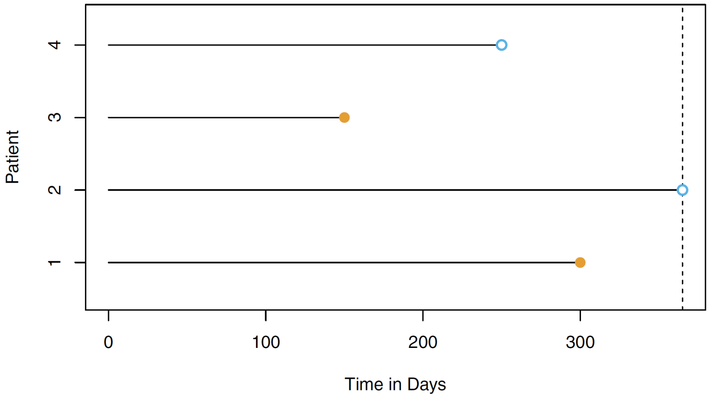
$$\delta = \begin{cases} 1 & \text{if } T \leq C, \\ 0 & \text{if } T > C. \end{cases}$$

- $\delta = 1$ if we observe the true survival time, and $\delta = 0$ if we instead observe the censoring time.
- We observe n (Y, δ) pairs,

$$(y_1, \delta_1), (y_2, \delta_2), \ldots, (y_n, \delta_n)$$

Example of Survival and Censoring Times

- We observe $n = 4$ patients for a 365-day follow-up period.
  - $(y_1, \delta_1) = (t_1 : 300, 1)$ and $(y_3, \delta_3) = (t_3 : 150, 1)$
  - $(y_2, \delta_2) = (c_2 : 365, 0)$ and $(y_4, \delta_4) = (c_4 : 260, 0)$



A Closer Look at Censoring

- We assume that the censoring mechanism is independent
    - Conditional on the features, the event time T is independent of the censoring time C.
- Data collection should be careful to determine whether independent censoring is a reasonable assumption.
- Type of censoring
    - Right censoring: $T \geq Y$, most events.
    - Left censoring: $T \leq Y$, e.g., pregnancy study.
    - Interval censoring: a censoring falls in some interval.
- Although left and interval censoring can be accommodated, most analysis for survival data assumes right censoring.

Survival Function

- The survival curve or survival function is defined as

$$S(t) = P(T > t)$$

- It quantifies the probability of surviving past time t.
- $S(t)$ means the probability that an event occurs later than t.
- The larger the value of $S(t)$, the less likely that the event will occur before time t.
- Estimation of $S(t)$ is complicated by the presence of censoring.
- Most popular approach to estimate $S(t)$ is Kaplan-Meier estimator.

Kaplan-Meier Estimator

- The K unique survival/censored times of patients are denoted by

$$d_1 < d_2 < \cdots < d_{K-1} < d_K$$

- q_k : the number of patients who died at time d_k.
- r_k : the number of patients alive just before d_k (at risk)
- The set of patients that are at risk at a given time are referred to as the risk set.
- It is natural to use the estimator

$$\hat{P}(T > d_j | T > d_{j-1}) = \frac{r_j - q_j}{r_j}$$

which is the fraction of the risk set at time d_j who survived past time d_j.

Kaplan-Meier Estimator

- By the law of total probability.

$$\begin{aligned} P(T > d_k) = & P(T > d_k | T > d_{k-1}) P(T > d_{k-1}) \\ & + P(T > d_k | T \leq d_{k-1}) P(T \leq d_{k-1}) \end{aligned}$$

- Since $P(T > d_k | T \leq d_{k-1}) = 0$,

$$S(d_k) = P(T > d_k) = P(T > d_k | T > d_{k-1}) P(T > d_{k-1}).$$

- Using the estimator above,

$$\begin{aligned} \hat{S}(d_k) &= \frac{r_j - q_j}{r_j} P(T > d_{k-1}) = \frac{r_j - q_j}{r_j} S(d_{k-1}) \\ &= \left(\frac{r_j - q_j}{r_j} \right) \cdot \left(\frac{r_{j-1} - q_{j-1}}{r_{j-1}} \right) S(d_{k-2}) \end{aligned}$$

Kaplan-Meier Estimator

- The Kaplan-Meier estimator of the survival curve

$$\hat{S}(d_k) = \prod_{j=1}^{k} \left(\frac{r_j - q_j}{r_j} \right)$$

- For times t between d_k and d_{k+1},

$$\hat{S}(t) = \hat{S}(d_k).$$

So, the Kaplan-Meier survival curve has a step-like shape.

Brain Cancer Data

- BrainCancer dataset contains the survival times for patients with primary brain tumors undergoing treatment with stereotactic radiation methods.
- The predictors are
  - gtv : gross tumor volume, in cubic centimeters
  - sex : male or female
  - diagnosis : meningioma, LG glioma, HG glioma, or other
  - loc : the tumor location(infratentorial or supratentorial)
  - ki : Karnofsky index
  - stereo : stereotactic method (SRS or SRT)
- Only 53 of the 88 patients were still alive at the end of the study, i.e.,

$$\sum_{i=1}^{88} \delta_i = 88 - 53 = 35$$

```
library(ISLR2)
data(BrainCancer)
```

```
?BrainCancer
names(BrainCancer)
BrainCancer
BrainCancer[ ,7:8]
```

```
lapply(BrainCancer[ ,-c(5, 8)], table)
summary(BrainCancer[ , c(5, 8)])
```

```
attach(BrainCancer)
plot(time, type="h", col=ifelse(status>0, "orange", "lightblue"),
     xlab="Patients", ylab="Survival time (Months)")
legend("topleft", c("Uncensored", "Censored"), lty=1,
       col=c("orange", "lightblue"))
```

```
d <- sort(unique(time))
q <- r <- NULL
for (i in 1:length(d)) {
    q[i] <- sum(time[status > 0] == d[i])
    r[i] <- sum(time >= d[i])
}
```

```
1-q/r
data.frame(d, q, r, S=cumprod(1-q/r))
```

```
library(survival)
Surv(time, status)
fit <- survfit(Surv(time, status) ~ 1)
data.frame(d=fit$time, q=fit$n.event, r=fit$n.risk, S=fit$surv)
```

```
plot(fit, col="blue", lty=1.5, xlab="Months",
     ylab="Estimated Probability of Survival")
```

Log-rank Test

- We wish to compare the survival of males to that of females.
- At first glance, a two-sample t-test seems like an obvious choice: we could test whether the mean survival time among the females equals the mean survival time among the males. But, the presence of censoring creates a complication.
- Alternatively, we can conduct a log-rank test, which examines how the events in each group unfold sequentially in time.
- The log-rank test is also known as the Mantel-Haenszel test or Cochran-Mantel-Haenszel test.
- Among the set of patients at risk at time d_k,

	Group 1	Group 2	Total
Died	q_{1k}	q_{2k}	q_k
Survived	$r_{1k} - q_{1k}$	$r_{2k} - q_{2k}$	$r_k - q_k$
Total	r_{1k}	r_{2k}	r_k

Log-rank Test

- The null hypothesis that there is no difference between the survival curves in the two groups.
- The log-rank test statistic can be computed by

$$W = \frac{\sum_{k=1}^{K} \left(q_{1k} - \mathrm{E}(q_{1k})\right)}{\sqrt{\sum_{k=1}^{K} \mathrm{Var}\left(q_{1k}\right)}} = \frac{\sum_{k=1}^{K} \left(q_{1k} - \frac{q_k}{r_k} r_{1k}\right)}{\sqrt{\sum_{k=1}^{K} \frac{q_k (r_{1k}/r_k)(1-r_{1k}/r_k)(r_k-q_k)}{r_k-1}}}$$

- When the sample size is large, the log-rank test statistic W has approximately a standard normal distribution.
- Alternatively, we can estimate the p-value via permutations, where we randomly swap the labels for the observations in the two groups in order to obtain an empirical distribution.
- The log-rank test is closely related to Cox's proportional hazards model with a single predictor.

```r
lapply(BrainCancer[,c("sex", "diagnosis", "stereo")], table)
```

```r
fit1 <- survfit(Surv(time, status) ~ sex)
plot(fit1, col=c(2, 4), lty=1.5, xlab="Months",
     ylab="Estimated Probability of Survival")
legend("topright", levels(sex), col=c(2, 4), lty=1.5)
survdiff(Surv(time, status) ~ sex)
```

```r
fit2 <- survfit(Surv(time, status) ~ stereo)
plot(fit2, col=c(2, 4), lty=1.5, xlab="Months",
     ylab="Estimated Probability of Survival")
legend("topright", levels(stereo), col=c(2, 4), lty=1.5)
survdiff(Surv(time, status) ~ stereo)
```

```r
fit3 <- survfit(Surv(time, status) ~ diagnosis)
plot(fit3, col=c(2,3,4,5), lty=1.5, xlab="Months",
     ylab="Estimated Probability of Survival")
legend("topright", levels(diagnosis), col=c(2,3,4,5), lty=1.5)
survdiff(Surv(time, status) ~ diagnosis)
```

Regression Models with a Survival Response

- We now consider the task of fitting a regression model to survival data
  - The response variable is the (possibly censored) survival time.
  - The predictor X is the p-dimensional vector.
- We wish to predict the true survival time T based on p features.
- Since the observed quantity Y is positive and may have a long right tail, we might be tempted to fit a linear regression of $\log(Y)$ on X.
- However, censoring again creates a problem since we are actually interested in predicting T and not Y.

Hazard Function

- The hazard function or hazard rate is formally defined as

$$h(t) = \lim_{\Delta t \to 0} \frac{P\left(t < T \leq t + \Delta t | T > t\right)}{\Delta t}$$

- It is the death rate in the instant after time t, given survival past that time.
- Higher values of $h(t)$ correspond to a higher probability of death.
- Since $P(A|B) = P(A \cap B)/P(B)$,

$$\begin{aligned} h(t) &= \lim_{\Delta t \to 0} \frac{P\left(\left(t < T \leq t + \Delta t\right) \cap \left(T > t\right)\right)/\Delta t}{P(T > t)} \\ &= \lim_{\Delta t \to 0} \frac{P\left(t < T \leq t + \Delta t\right)/\Delta t}{P(T > t)} = \frac{f(t)}{S(t)} \end{aligned}$$

Hazard Function

- The probability density function associated with T is

$$f(t) = \lim_{\Delta t \to 0} \frac{P\left(t < T \leq t + \Delta t\right)}{\Delta t}$$

- It is the instantaneous rate of death at time t.
- The cumulative hazard function is defined as

$$H(t) = \int_0^t h(y)dy.$$

  Consequently, $S(t) = e^{-H(t)}$.
- Note that the functions $h(t)$ and $H(t)$ are not a probability functions. But, both functions measure a risk.

Hazard Function

- In order to perform a regression with a survival time and p features, we need is model the survival time as a function of the p covariates (predictors).
- One possible approach is to assume a functional form for the hazard function such as

$$h(t|x_i) = \exp\left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}\right),$$

  where the exponential function guarantees that the hazard function is non-negative.
- However, this approach is quite restrictive, in the sense that it requires us to make a very stringent assumption on the form of the hazard function.

Relative Risk

- The relative risk of a patient who has observed levels $x_i = (x_{i1}, \cdots, x_{ip})$ (relative to a subject with each explanatory variable equal to 0) as:

$$RR(t|x_i) = \frac{h(t|x_i)}{h(t|x_i = (0, \ldots, 0))} = \frac{h(t|x_i)}{h_0(t)}$$

- The relative risk is the ratio of the probability of event for one group relative to another.
- One common model for the relative risk is to assume that it is constant over time,

$$RR(t|x_i) = \exp\left(\sum_{j=1}^{p} \beta_j x_{ij}\right)$$

Proportional Hazards

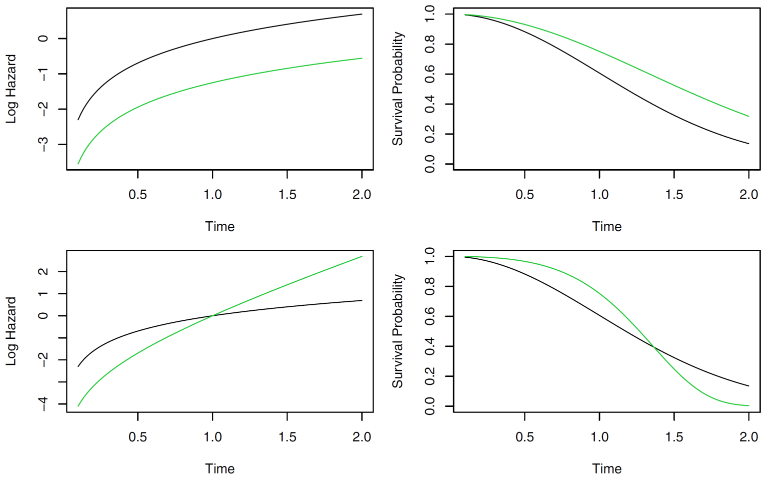
- The proportional hazards assumption states that

$$h(t|x_i) = h_0(t) \exp\left(\sum_{j=1}^{p} \beta_j x_{ij}\right),$$

  where $h_0(t) \geq 0$ is an unspecified function, known as the baseline hazard.
- $h_0(t)$ is the hazard function for an individual with features $x_{i1} = \ldots = x_{ip} = 0$.
- Since we make no assumptions about its functional form, we allow the hazard function to take any form.
- Therefore, the hazard function is very flexible and can model a wide range of relationships between the covariates and survival time.

Example of Proportional Hazards

- A simple example with $p = 1$ and $x_i \in \{0, 1\}$.



Cox's Proportional Hazards Model

- The Cox's proportional hazards model estimates $\beta = (\beta_1, \ldots, \beta_p)$ without having to specify the form of $h_0(t)$.
- It assumes that there are no ties among the failure, or death, times, i.e., each failure occurs at a distinct time.
- The total hazard at time y_i for the at risk observations is

$$\sum_{i':y_{i'} \geq y_i} h_0(y_i) \exp\left(\sum_{j=1}^{p} \beta_j x_{i'j}\right)$$

- The "at risk" observations at time y_i are those that are still at risk of failure, i.e. those that have not yet failed or been censored before time y_i.

Cox's Proportional Hazards Model

- The probability that the ith observation fails at time y_i is

$$\frac{h_0(y_i)\exp\left(\sum_{j=1}^{p}\beta_j x_{ij}\right)}{\sum_{i':y_{i'}\geq y_i} h_0(y_i)\exp\left(\sum_{j=1}^{p}\beta_j x_{i'j}\right)}.$$

- The partial likelihood is simply the product of these probabilities over all of the uncensored observations,

$$L(\beta)=\prod_{i:\delta_i=1}\frac{\exp\left(\sum_{j=1}^{p}\beta_j x_{ij}\right)}{\sum_{i':y_{i'}\geq y_i}\exp\left(\sum_{j=1}^{p}\beta_j x_{i'j}\right)}$$

- To estimate β, we simply maximize the partial likelihood with respect to β.

Cox's Proportional Hazards Model

- Suppose we have just a single predictor $(p = 1)$ with $x_i \in \{0, 1\}$.
- We want to determine whether there is a difference between the survival times of the observations in the group $(x_i = 0)$ and those in the group $(x_i = 1)$.
  - Approach 1: Fit a Cox's proportional hazards model, and test $H_0 : \beta = 0$.
  - Approach 2: Perform a log-rank test to compare the two groups.
- Approach 1 can be conducted by the likelihood ratio test, Wald test and score test.
- There is a close relationship between these two approaches.
- The score test for $H_0 : \beta = 0$ in Cox's proportional hazards model is exactly equal to the log-rank test.

Cox's Proportional Hazards Model

- There is no intercept in the Cox's proportional hazard model. The intercept can be absorbed into the baseline hazard $h_0(t)$.
- We have assumed that there are no tied failure times. In the case of ties, the exact form of the partial likelihood is a bit more complicated.
- The partial likelihood is not exactly a likelihood. So, it does not correspond exactly to the probability of the data, even if the former is a very good approximation to the later.
- We have focused only on estimation of β. However, at times we may also wish to estimate the baseline hazard $h_0(t)$ to obtain the survival curve $S(t|x)$ for an individual with feature vector x.

```r
library(survival)
data(BrainCancer, package="ISLR2")
attach(BrainCancer)
```

```r
fit.sex <- coxph(Surv(time, status) ~ sex)
summary(fit.sex)
```

```r
fit.all <- coxph(Surv(time, status) ~  sex + diagnosis + loc
                 + ki + gtv + stereo)
fit.all
summary(fit.all)
```

```r
TT <- data.frame(diagnosis=levels(diagnosis), sex=rep("Female",4),
                 loc=rep("Supratentorial",4), ki=rep(mean(ki),4),
                 gtv=rep(mean(gtv),4), stereo=rep("SRT",4))
SS <- survfit(fit.all, newdata=TT)
plot(SS, xlab="Months", ylab="Survival Probability", col=2:5)
legend("bottomleft", levels(diagnosis), col=2:5, lty=1.5)
```

Shrinkage for the Cox Model

- For analysis of high-dimensional data ($p \gg n$), the Cox's model can be applied using the penalty function.
- The penalized log-likelihood is

$$-\log\left(\prod_{i:\delta_i=1} \frac{\exp\left(\sum_{j=1}^{p} \beta_j x_{ij}\right)}{\sum_{i':y_{i'} \geq y_i} \exp\left(\sum_{j=1}^{p} \beta_j x_{i'j}\right)}\right) + \lambda P(\beta),$$

  where λ is a tuning parameter to control sparsity and $P(\cdot)$ is a penalty function.
- When λ is large, the lasso penalty will give some coefficients that are exactly equal to zero.
- We can select features which have nonzero regression coefficients, since they are likely to associated with the survival time.

```
library(glmnet)
sum(is.na(BrainCancer))
BC <- na.omit(BrainCancer)
x0 <- model.matrix(~., BC)[,-c(1,10,11)]
y <- Surv(BC$time, BC$status)
```

```
fun <- function(t) sqrt(var(t)*(length(t)-1)/length(t))
sdx <- matrix(apply(x0, 2, fun), dim(x0)[2], dim(x0)[1])
x <- x0/t(sdx)
```

```
g <- glmnet(x, y, alpha=1, family="cox")
par(mfrow=c(1,2))
plot(g, "lambda", label=TRUE)
plot(g, "norm", label=TRUE)
```

```
set.seed(1234)
gcv <- cv.glmnet(x, y, alpha=1, family="cox", nfolds=5)
dev.off()
plot(gcv)
coef(gcv, s=c(gcv$lambda.min, gcv$lambda.1se))
```

Time-Dependent Covariates

- The proportional hazards model can handle time-dependent covariates, whose value may change over time.
- For example, a patient's blood pressure every week, so x_i should be replaced by $x_i(t)$.
- One example of time-dependent covariates appears in the analysis of data from a heart transplant program.
  - Patients in need of a transplant were put on a waiting list.
  - Some patients received a transplant, but others died while still on the waiting list.
  - Can we determine whether a transplant was associated with longer patient survival?
  - Since patients had to live long enough to get a transplant, on average, healthier patients received transplants.
  - The problem can be solved by using a time-dependent covariate for transplant.