02. Linear Model Selection and Regularization

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.,

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

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.

```
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]
    }
}
```

```
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)
```

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

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

  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)

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

  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.

Model Selection Criteria

```r
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)
```

```r
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)
```

Best Model Selection

```r
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))
```

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

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

```r
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.

```r
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] <- mean((Hitters$Salary[test]-pred)^2)
}
val.errors
w <- which.min(val.errors)
```

```r
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)
```

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

```
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] <- mean((Hitters$Salary[tt]-pred)^2)
    }
}
```

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

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

```
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")
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)
```

Variable Selection Methods

- Best subset selection method has a limitation to apply for high-dimensional data due to dimensionality.
- Alternatively, forward and backward selection methods can be applied, but they are not guaranteed to find the best possible model out of all 2^p models containing subsets of the p predictors.
- Particularly, 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 selection can be used even when $n < p$, and so is the only viable subset method when p is very large.
- For analysis of high-dimensional data, a shrinkage method is recommended.

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

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

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

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

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

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

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

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

```r
library(glmnet)
x <- model.matrix(Salary~., Hitters)[, -1]
y <- Hitters$Salary
```

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

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

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

```r
plot(ridge.mod, "lambda")
```

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.

```r
lasso.mod <- glmnet(x, y, alpha=1)
dim(coef(lasso.mod))
las <- cbind(lasso.mod$lambda, lasso.mod$df)
colnames(las) <- c("lambda", "df")
las
```

```r
dim(lasso.mod$beta)
apply(lasso.mod$beta, 2, function(t) sum(t!=0))
apply(lasso.mod$beta, 2, function(t) sum(abs(t)))
```

```r
l1.norm <- apply(lasso.mod$beta, 2, function(t) sum(abs(t)))
x.axis <- cbind(log(lasso.mod$lambda), l1.norm)
colnames(x.axis) <- c("log.lambda", "L1.norm")
x.axis
```

```r
par(mfrow=c(1,2))
plot(lasso.mod, "lambda", label=TRUE)
plot(lasso.mod, "norm", label=TRUE)
```

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

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

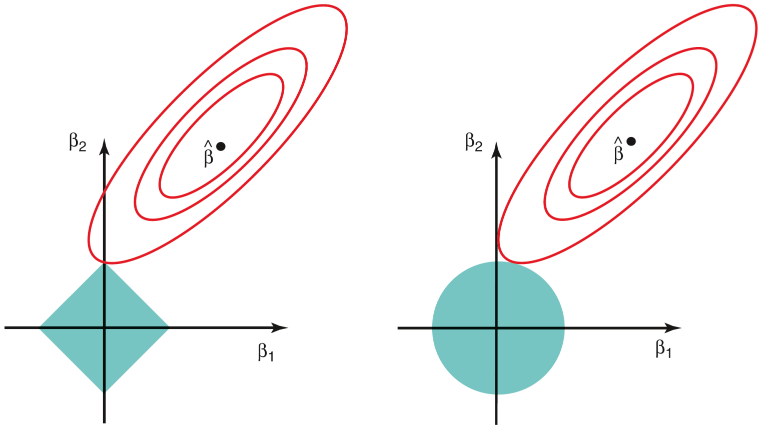
and

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

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.

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

- 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)
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(sum(b.min[-1]^2), sum(b.1se[-1]^2))
```

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

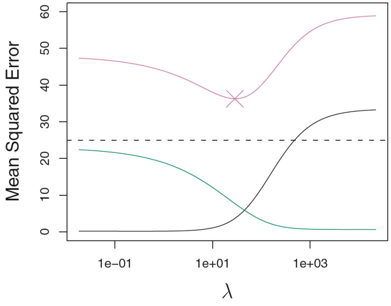
```
log(cv.l$lambda.min)
log(cv.l$lambda.1se)
```

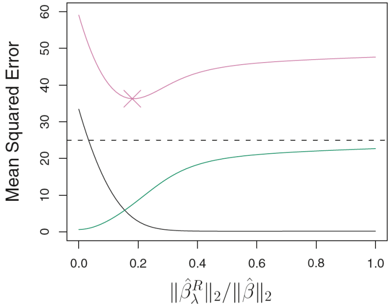
```
which(cv.l$lambda==cv.l$lambda.min)
which(cv.l$lambda==cv.l$lambda.1se)
```

```
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(b.min!=0)-1, sum(b.1se!=0)-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.

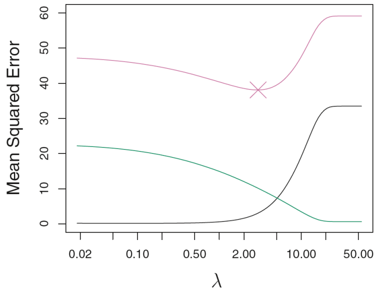
```
set.seed(1234)
K <- 100; p <- 45; n <- 50
beta <- runif(p, -1, 1)
lam <- 10^seq(3, -3, -0.1)
```

```
RES <- array(0, c(n, length(lam), K))
x <- matrix(rnorm(n * p), n, p)
y <- x %*% beta + rnorm(n, 0, 0.5)
for (i in 1:K) {
    gr <- sample(rep(seq(5), length=length(y)))
    yhat <- array(0, c(n, length(lam)))
    for (j in 1:5) {
        tran <- gr!=j
        g <- glmnet(x[tran,], y[tran], alpha=0, lambda=lam)
        wh <- which(gr==j)
        xbeta <- cbind(1, x[wh,]) %*% coef(g)
        yhat[wh, ] <- as.matrix(xbeta)
    }
    RES[,,i] <- yhat
}
```

```
MSE0 <- Bias0 <- Vars0 <- matrix(0, n, length(lam))
for (k in 1:length(lam)) {
    PE <- (RES[,k,] - matrix(y, n, K))^2
    MSE0[,k] <- apply(PE, 1, mean)
    Bias0[,k] <- abs(apply(RES[,k,], 1, mean) - y)
    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)
rlam <- rev(lam)
matplot(MAT, type="l", col=c(1,3,2), lty=1, xaxt="n",
        xlab=expression(lambda), ylab="Mean Squared Error")
legend("topleft", c("Bias", "Variance", "MSE"),
       col=c(1,3,2), lty=1)
w <- which.min(MAT[,3])
points(w, MAT[w, 3], pch=4, col="red", cex=2)
cc <- c(1, seq(11, 61, 10))
axis(1, at=cc, labels=round(rlam[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.

```r
set.seed(12345)
K <- 100; p <- 45; n <- 50
beta <- rep(0, p)
beta[1:10] <- runif(10, -1, 1)
lam <- 10^seq(0, -3, -0.05)
```

```r
RES <- array(0, c(n, length(lam), K))
x <- matrix(rnorm(n * p), n, p)
y <- x %*% beta + rnorm(n, 0, 0.5)
for (i in 1:K) {
    gr <- sample(rep(seq(5), length=length(y)))
    yhat <- array(0, c(n, length(lam)))
    for (j in 1:5) {
        tran <- gr!=j
        g <- glmnet(x[tran,], y[tran], alpha=1, lambda=lam)
        wh <- which(gr==j)
        xbeta <- cbind(1, x[wh,]) %*% coef(g)
        yhat[wh, ] <- as.matrix(xbeta)
    }
    RES[,,i] <- yhat
}
```

```r
MSE1 <- Bias1 <- Var1 <- matrix(0, n, length(lam))
for (k in 1:length(lam)) {
    PE1 <- (RES[,k,] - matrix(y, n, K))^2
    MSE1[,k] <- apply(PE1, 1, mean)
    Bias1[,k] <- abs(apply(RES[,k,], 1, mean) - y)
    Var1[,k] <- apply(RES[,k,], 1, var)
}
MSE <- apply(MSE1, 2, mean)
Bias <- apply(Bias1, 2, mean)
Vars <- apply(Var1, 2, mean)
```

```r
MAT <- apply(cbind(Bias^2, Vars, MSE), 2, rev)
rlam <- rev(lam)
matplot(MAT, type="l", col=c(1,3,2), lty=1, xaxt="n",
        xlab=expression(lambda), ylab="Mean Squared Error")
legend("topleft", c("Bias", "Variance", "MSE"),
       col=c(1,3,2), lty=1)
w <- which.min(MAT[,3])
points(w, MAT[w, 3], pch=4, col="red", cex=2)
cc <- c(1, 21, 41, 61)
axis(1, at=cc, labels=rlam[cc])
```

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.

Lasso vs. Ridge

```
MSE.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)
    }
    MSE0 <- Bias0 <- Vars0 <-  array(0, c(n,length(lam)))
    MSE1 <- Bias1 <- Vars1 <-  array(0, c(n,length(lam)))
    for (j in 1:length(lam)) {
        PE0 <-(yhat0[,j,] - matrix(ytest, n, K))^2
        PE1 <-(yhat1[,j,] - matrix(ytest, n, K))^2
        MSE0[ ,j] <- apply(PE0, 1, mean)
        MSE1[ ,j] <- apply(PE1, 1, mean)
        BS0 <- abs(yhat0[,j,] - matrix(ytest, n, K))
        BS1 <- abs(yhat1[,j,] - matrix(ytest, n, K))
### The function "MSE.fun" continues
        Bias0[,j] <- apply(BS0, 1, mean)
        Bias1[,j] <- apply(BS1, 1, mean)
        Vars0[,j] <- apply(yhat0[,j,], 1, var)
        Vars1[,j] <- apply(yhat1[,j,], 1, var)
    }
    MSE.r <- apply(MSE0, 2, mean)
    MSE.l <- apply(MSE1, 2, mean)
    Bia.r <- apply(Bias0, 2, mean)
    Bia.l <- apply(Bias1, 2, mean)
    Var.r <- apply(Vars0, 2, mean)
    Var.l <- apply(Vars1, 2, mean)

    ridge <- apply(cbind(Bia.r^2, Var.r, MSE.r), 2, rev)
    lasso <- apply(cbind(Bia.l^2, Var.l, MSE.l), 2, rev)
    newlam <- rev(lam)
list(ridge=ridge,lasso=lasso, lambda=newlam)
}
```

```
set.seed(111000)
K <- 10
p <- 120
n <- 100
lam <- 10^seq(1, -3, -0.05)
x.test <- matrix(rnorm(n * p), n, p)
```

```
## The case that all predictors have non-zero coefficients
beta1 <- beta2 <- runif(p, -1, 1)
ytest1 <- x.test %*% beta1 + rnorm(n)
g1 <- MSE.fun(n, p, K, beta1, lam, xtest, ytest1)
RES1 <- cbind(g1$lasso, g1$ridge)
```

```
## The case that only 5 predictors have non-zero coefficients
beta2[6:p] <- 0
ytest2 <- x.test %*% beta2 + rnorm(n)
g2 <- MSE.fun(n, p, K, beta2, lam, xtest, ytest2)
RES2 <- cbind(g2$lasso, g2$ridge)
```

```
par(mfrow=c(1,2))
matplot(RES1, type="l", col=c(1,3,2), lty=rep(1:2,each=3),
        xlab=expression(lambda), xaxt="n",
        ylab="Mean Squared Error")
cc <- c(1, seq(21, 81, 20))
axis(1, at=cc, labels=g1$lambda[cc])
```

```
matplot(RES2, type="l", col=c(1,3,2), lty=rep(1:2,each=3),
        xlab=expression(lambda), xaxt="n",
        ylab="Mean Squared Error")
legend("topright",c("Bias_Lasso", "Variance_Lasso", "MSE_Lasso",
       "Bias_Ridge", "Variance_Ridge", "MSE_Ridge"),
       col=c(1,3,2), lty=rep(1:2, each=3))
axis(1, at=cc, labels=g2$lambda[cc])
```

Regularization Methods

- Regularization methods are based on a penalized likelihood

$$Q_\lambda(\beta_0, \boldsymbol{\beta}) = -l(\beta_0, \boldsymbol{\beta}) + p_\lambda(\boldsymbol{\beta})$$

- $l(\beta_0, \boldsymbol{\beta})$ is a log-likelihood function of $\boldsymbol{\beta} = \{\beta_1, \ldots, \beta_p\}$, which is the p-dimensional vector of regression coefficients.
- $l(\beta_0, \boldsymbol{\beta})$ is determined by a type of the response variable.
- $p_\lambda(\boldsymbol{\beta})$ is a penalty function, where a tuning parameter λ controls sparsity.
- Solution to regression coefficients can be obtained by

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \arg\min_{\beta_0, \boldsymbol{\beta}} Q_\lambda(\beta_0, \boldsymbol{\beta}),$$

for a fixed value of λ.

Penalized Likelihood for Quantitative Outcome

- For the i-th individual,
  - y_i : a quantitative outcome
  - $x_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}}$: the p-dimensional predictor.
- Linear regression model can be defined as

$$y_i = \beta_0 + x_i^{\mathrm{T}} \boldsymbol{\beta} + \epsilon_i,$$

  where $\epsilon \sim N(0, \sigma^2)$.
- Least squared loss is used for the negative log-likelihood

$$\begin{aligned} Q_\lambda(\beta_0, \boldsymbol{\beta}) &= -l(\beta_0, \boldsymbol{\beta}) + p_\lambda(\boldsymbol{\beta}) \\ &= \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \beta_0 - x_i^{\mathrm{T}} \boldsymbol{\beta} \right)^2 + p_\lambda(\boldsymbol{\beta}) \end{aligned}$$

Penalized Likelihood for Binary Outcome

- For the i-th individual,
  - y_i : a binary outcome ($y_i = 1$ for cases and $y_i = 0$ for controls)
- In case-control studies, the penalized likelihood is

$$Q_\lambda(\beta_0, \boldsymbol{\beta}) = -l(\beta_0, \boldsymbol{\beta}) + p_\lambda(\boldsymbol{\beta}).$$

- A logistic likelihood $l(\beta_0, \boldsymbol{\beta})$ is

$$= \sum_{i=1}^{n} \left[y_i \log p_i(\beta_0, \boldsymbol{\beta}) + (1 - y_i) \log(1 - p_i(\beta_0, \boldsymbol{\beta})) \right],$$

where

$$p_i(\beta_0, \boldsymbol{\beta}) = \frac{e^{\beta_0 + x_i^{\mathrm{T}} \boldsymbol{\beta}}}{1 + e^{\beta_0 + x_i^{\mathrm{T}} \boldsymbol{\beta}}}.$$

CV Error for Binary Outcomes

- Suppose that $y_i = 1$ for a case and $y_i = 0$ for a control.
- CVE based on deviance

$$\text{CVE} = \frac{1}{n}\sum_{k=1}^{K} -2 \sum_{i\in C_k} \left(y_i \log \hat{p}_i^{[-k]} + (1-y_i)\log(1-\hat{p}_i^{[-k]}) \right),$$

  where $\hat{p}_i^{[-k]}$ is the probability of $y_i = 1$ for the observation i, obtained from the data with part k removed.
- CVE based on classification error

$$\text{CVE} = \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in C_k} I\left(y_i - \hat{y}_i^{[-k]} \right),$$

  where $\hat{y}_i^{[-k]}$ is the fit for the observation i, obtained from the data with part k removed.

Example: Heart Data

- These data contain a binary outcome `AHD` for 303 patients who presented with chest pain.
- An outcome value of `Yes` indicates the presence of heart disease based on an angiographic test, while No means no heart disease.
- There are 13 predictors including `Age`, `Sex`, `Chol` (a cholesterol measurement), and other heart and lung function measurements.
- We can find some missing values from data, so we need to remove the samples with missing values. Then, the sample size is reduced down to 297.

```
url.ht <- "https://www.statlearning.com/s/Heart.csv"
Heart <- read.csv(url.ht, h=T)
summary(Heart)
```

```
Heart <- Heart[, colnames(Heart)!="X"]
Heart[,"Sex"] <- factor(Heart[,"Sex"], 0:1, c("female", "male"))
Heart[,"Fbs"] <- factor(Heart[,"Fbs"], 0:1, c("false", "true"))
Heart[,"ExAng"] <- factor(Heart[,"ExAng"], 0:1, c("no", "yes"))
Heart[,"ChestPain"] <- as.factor(Heart[,"ChestPain"])
Heart[,"Thal"] <- as.factor(Heart[,"Thal"])
Heart[,"AHD"] <- as.factor(Heart[,"AHD"])
```

```
summary(Heart)
dim(Heart)
sum(is.na(Heart))
```

```
Heart <- na.omit(Heart)
dim(Heart)
summary(Heart)
```

```
## Logistic Regression
g <- glm(AHD ~., family="binomial", Heart)
summary(g)
```

```
x <- model.matrix(AHD ~., Heart)[,-1]
y <- Heart$AHD
g1 <- glmnet(x, y, family="binomial")
```

```
par(mfrow=c(1,2))
plot(g1, "lambda", label=TRUE)
plot(g1, "norm", label=TRUE)
```

```
df <- cbind(g1$lambda, g1$df)
colnames(df) <- c("lambda", "nonzero")
rownames(df) <- colnames(g1$beta)
df
```

```
g1$beta[, c("s0", "s5", "s15", "s25", "s35")]
coef(g1, s=g1$lambda[16])
```

```
set.seed(1234)
K <- 10
n <- length(y)
fold <- sample(rep(1:K, length.out=n))
lam <- g1$lambda
```

```
MSE <- matrix(0, n, length(lam))
for (i in 1:K) {
    test <- fold==i
    tran <- fold!=i
    g <- glmnet(x[tran,], y[tran], family="binomial", lambda=lam)
    prob <- predict(g, x[test,], s=lam, type="response")
    yval <- y[test]=="Yes"
    MSE[test, ] <- -2*(yval*log(prob) + (1-yval)*log(1-prob))
}
CVE <- apply(MSE, 2, mean)
```

```
lam[which.min(CVE)]
coef(g, s=lam[which.min(CVE)])
g1 <- glmnet(x, y, family="binomial", lambda=lam[which.min(CVE)])
coef(g1)
```

```
par(mfrow=c(1,2))
plot(log(lam), CVE, type="b", xlab="log lambda", ylab="CV errors")
abline(v=log(lam)[which.min(CVE)], col="red", lty=2)
```

```
set.seed(123)
R <- 8; CVE <- matrix(0, R, length(lam))
for (j in 1:R) {
    fold <- sample(rep(1:K, length.out=n))
    MSE <- matrix(0, n, length(lam))
    for (i in 1:K) {
        test <- fold==i
        tran <- fold!=i
        g <- glmnet(x[tran,], y[tran], family="binomial",
                    lambda=lam)
        prob <- predict(g, x[test,], s=lam, type="response")
        yval <- y[test]=="Yes"
        MSE[test, ] <- -2*(yval*log(prob) + (1-yval)*log(1-prob))
    }
CVE[j,] <- apply(MSE, 2, mean)
}
matplot(log(lam), t(CVE), type="l", xlab=expression(log(lambda)),
      ylab="Deviance", main="10-fold CV")
```

```
set.seed(123)
par(mfrow=c(1,3))
gcv1 <- cv.glmnet(x, y, family="binomial", lambda=lam,
                  nfolds=5, type.measure="deviance")
plot(gcv1)
```

```
gcv2 <- cv.glmnet(x, y, family="binomial", lambda=lam,
                  nfolds=5, type.measure="class")
plot(gcv2)
```

```
gcv3 <- cv.glmnet(x, y, family="binomial", lambda=lam,
                  nfolds=5, type.measure="auc")
plot(gcv3)
```

```
c(gcv1$lambda.min, gcv1$lambda.1se)
c(gcv2$lambda.min, gcv2$lambda.1se)
c(gcv3$lambda.min, gcv3$lambda.1se)
```

```
g0 <- glmnet(x, y, family="binomial", lambda=lam)
coef(g0, s=c(gcv1$lambda.min, gcv2$lambda.min, gcv3$lambda.min))
coef(g0, s=c(gcv1$lambda.1se, gcv2$lambda.1se, gcv3$lambda.1se))
```

Simulation Study: Prediction Performance

```r
## Generate X matrix and Y vector
sim.fun <- function(n, p, beta, family=c("gaussian", "binomial"))
{
    family <- match.arg(family)
    if (family=="gaussian") {
         x <- matrix(rnorm(n*p), n, p)
         y <- x %*% beta + rnorm(n)
    }
    else {
         x <- matrix(rnorm(n*p), n, p)
         xb <- x %*% beta
         z <- exp(xb) / (1+exp(xb))
         u <- runif(n)
         y <- rep(0, n)
         y[z > u] <- 1
    }
list(x=x, y=y)
}
```

Simulation Study: Quantitative Outcome

```
set.seed(1234)
n <- 200
p <- 2000
beta <- rep(0, p)
beta[1:20] <- runif(20, -1, 1)
```

```
sim <- sim.fun(n, p, beta)
x <- sim$x; y <- sim$y
```

```
## Fit the lasso with two different lambda values
g <- cv.glmnet(x, y, alpha=1, nfolds = 10)
bhat1 <- coef(g, s="lambda.min")
bhat2 <- coef(g, s="lambda.1se")
wh1 <- which(as.matrix(bhat1)!=0)
w1 <- wh1[-1]-1
wh2 <- which(as.matrix(bhat2)!=0)
w2 <- wh2[-1]-1
```

```
## Compute ordinary least square estimates (unbiased estimates)
bhat3 <- bhat4 <- bhat5 <- rep(0, p+1)
bhat3[wh1] <- lm(y ~ x[, w1])$coef
bhat4[wh2] <- lm(y ~ x[, w2])$coef
bhat5[1:21] <- lm(y ~ x[, 1:20])$coef
```

```
set.seed(56789)
```

```
## Generate test sets
test <- sim.fun(n, p, beta)
xt <- cbind(1, test$x)
yt <- test$y
```

```
## Test set prediction errors of 6 coefficient estimates
mean((yt - xt %*% bhat1)^2) # lasso_lambda.min (M1)
mean((yt - xt %*% bhat2)^2) # lasso_lambda.1se (M2)
mean((yt - xt %*% bhat3)^2) # least square_lambda.min (M3)
mean((yt - xt %*% bhat4)^2) # least square_lambda.1se (M4)
mean((yt - xt %*% bhat5)^2) # least square_nonzero beta (M5)
mean((yt - xt %*% c(0, beta))^2) # true beta (M6)
```

```
set.seed(1)
```

```
## Generate new test sets 100 times
K <- 100
pred <- matrix(NA, K, 6)
```

```
for (i in 1:K) {
    test <- sim.fun(n, p, beta)
    xt <- cbind(1, test$x)
    yt <- test$y
    pred[i, 1] <- mean((yt - xt %*% bhat1)^2)
    pred[i, 2] <- mean((yt - xt %*% bhat2)^2)
    pred[i, 3] <- mean((yt - xt %*% bhat3)^2)
    pred[i, 4] <- mean((yt - xt %*% bhat4)^2)
    pred[i, 5] <- mean((yt - xt %*% bhat5)^2)
    pred[i, 6] <- mean((yt - xt %*% c(0, beta))^2)
}
```

```
apply(pred, 2, mean)
boxplot(pred, col=c(2,2,4,4,3,"orange"), boxwex=0.6,
        names=c("M1", "M2", "M3", "M4", "M5", "M6"),
        ylab="Prediction Error")
```

Simulation Study: Binary Outcome

```
set.seed(111)
n <- 200
p <- 2000
beta <- rep(0, p)
beta[1:20] <- runif(20, -1, 1)
sim <- sim.fun(n, p, beta, family="binomial")
x <- sim$x; y <- sim$y
```

```
g <- cv.glmnet(x, y, alpha=1, nfolds = 10, family="binomial")
bhat1 <- coef(g, s="lambda.min")
bhat2 <- coef(g, s="lambda.1se")
wh1 <- which(as.matrix(bhat1)!=0)
wh2 <- which(as.matrix(bhat2)!=0)
```

```
bhat3 <- bhat4 <- bhat5 <- rep(0, p+1)
w1 <- wh1[-1]-1; w2 <- wh2[-1]-1
bhat3[wh1] <- glm(y ~ x[, w1], family="binomial")$coef
bhat4[wh2] <- glm(y ~ x[, w2], family="binomial")$coef
bhat5[1:21] <- glm(y ~ x[, 1:20], family="binomial")$coef
```

```
## Classification Error
class.fun <- function(test.x, test.y, beta, k=0.5) {
  xb <- test.x %*% beta
  exb <- exp(xb) / (1 + exp(xb))
  y <- rep(0, length(test.y))
  y[as.logical(exb > k)] <- 1
min(mean(test.y!=y), mean(test.y!=(1-y)))
}
```

```
## Generate test sets
set.seed(56789)
test <- sim.fun(n, p, beta, family="binomial")
xt <- cbind(1, test$x); yt <- test$y
```

```
## Prediction error comparison
class.fun(xt, yt, bhat1)   # lasso_lambda.min  (M1)
class.fun(xt, yt, bhat2)   # lasso_lambda.1se  (M2)
class.fun(xt, yt, bhat3)   # least square_lambda.min (M3)
class.fun(xt, yt, bhat4)   # least square_lambda.1se (M4)
class.fun(xt, yt, bhat5)   # least square_nonzero beta (M5)
class.fun(xt, yt, c(0, beta))    # true beta (M6)
```

```
set.seed(35791)
```

```
## Generate new test sets 100 times
K <- 100
pred <- matrix(NA, K, 6)
```

```
for (i in 1:K) {
    test <- sim.fun(n, p, beta, family="binomial")
    xt <- cbind(1, test$x)
    yt <- test$y
    pred[i, 1] <- class.fun(xt, yt, bhat1)
    pred[i, 2] <- class.fun(xt, yt, bhat2)
    pred[i, 3] <- class.fun(xt, yt, bhat3)
    pred[i, 4] <- class.fun(xt, yt, bhat4)
    pred[i, 5] <- class.fun(xt, yt, bhat5)
    pred[i, 6] <- class.fun(xt, yt, c(0, beta))
}
apply(pred, 2, mean)
```

```
boxplot(pred, col=c(2,2,4,4,3,"orange"), boxwex=0.6,
        names=c("M1", "M2", "M3", "M4", "M5", "M6"),
        ylab="Prediction Error")
```

Regularization Methods

- Regularization methods consists of the negative log-likelihood and a penalty function

$$Q_\lambda(\beta_0, \boldsymbol{\beta}) = -l(\beta_0, \boldsymbol{\beta}) + p_\lambda(\boldsymbol{\beta})$$

- The negative log-likelihood function can be defined as
  - Quantitative outcome: least square loss function
  - Binary outcome: logistic likelihood
  - Matched case-control outcome: conditional logistic likelihood
  - Count outcome: Poisson likelihood
  - Qualitative outcome: Multinomial likelihood
  - Survival outcome: Cox partial likelihood

Types of Penalty Functions

- Convex penalty functions
  - Lasso (Tibshirani, JRSS, 1996)
  - Fused lasso (Tibshirani *et al.* JRSS, 2005)
  - Adaptive lasso (Zou, JASA, 2006)
  - Elastic-net (Zou and Hastie, JRSS, 2005)
- Non-convex penalty functions
  - l_q-norm penalty with $0 < q < 1$
  - Smoothly clipped absolute deviation (SCAD) (Fan and Li, JASA, 2005)
  - Minimax concave penalty (MCP) (Zhang, AOS, 2010)
- Group structure penalty functions
  - Group lasso (Yuan and Lin, JRSS, 2006)
  - Graph-constrained regularization (Li and Li, AOAS 2010)
  - Sparse group lasso (Simon *et al.* JGCS 2013)

Lasso Regression and Ridge Regression

- Lasso regression uses a l_1-norm penalty

$$p_\lambda(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|_1 = \lambda\sum_{j=1}^{p}|\beta_j|.$$

- Lasso can perform variable selection since
  - $\hat{\beta}_j = 0$ for most of $j \in \{1, 2, \dots, p\}$.
- Ridge regression uses a squared l_2-norm penalty

$$p_\lambda(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|_2^2 = \lambda\sum_{j=1}^{p}\beta_j^2$$

- Ridge cannot perform variable selection since
  - $\hat{\beta}_j \neq 0$ for all $j \in \{1, 2, \dots, p\}$.

The Limitations of Lasso and Ridge

- Lasso penalty function (l_1-norm)
  - If $p > n$, lasso selects at most n variables. The number of selected variables is bounded by the number of samples.
  - Lasso is indifferent to highly correlated variables and tends to pick only one variable and ignore the rest.
- Ridge penalty function (squared l_2-norm)
  - It cannot perform variable selection. (No coefficients are zero.)
  - But, ridge regression shrinks correlated features to each other
- Elastic-net regularization
  - The elastic-net combines the l_1 and squared l_2 penalties
  - The l_1 part of the penalty generates a sparse model.
  - The l_2 part of the penalty encourages grouping effects.

Elastic-Net Regularization

- The penalty function of elastic-net is

$$\begin{aligned} p_\lambda(\boldsymbol{\beta}) &= \lambda\left(\alpha\|\boldsymbol{\beta}\|_1 + (1-\alpha)\|\boldsymbol{\beta}\|_2^2\right) \\ &= \lambda\alpha\sum_{j=1}^{p}|\beta_j| + \lambda(1-\alpha)\sum_{j=1}^{p}\beta_j^2 \end{aligned}$$

  where $\alpha \in [0,1]$ is a mixing proportion between lasso and ridge.
  - Lasso if $\alpha = 1$
  - Ridge if $\alpha = 0$.
- Convex penalty function
  - Efficient computation (ideal for high-dimensional data).
  - Different computational procedures lead to the same solutions.
  - Relatively many convex optimization algorithms available

Elastic-Net Regularization

- Minimize the penalized log-likelihood,

$$\hat{\boldsymbol{\beta}}_{\lambda,\alpha} = \arg\min_{\boldsymbol{\beta}} \left[-l(\boldsymbol{\beta}) + \lambda\alpha\|\boldsymbol{\beta}\|_1 + \lambda(1-\alpha)\|\boldsymbol{\beta}\|_2^2 \right],$$

  where tuning parameters α and λ are fixed.
- Cyclic coordinate descent algorithm
  - Efficient and fast convergence for even large p
  - Given α, the solution $\hat{\boldsymbol{\beta}}$ can be obtained for a grid of λ values,

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

    for $\lambda_1 > \lambda_2 > \ldots, > \lambda_{m-1} > \lambda_m$.
- Need to pick up the optimal λ and α.

Application to the Golub data

- The gene expression data collected by Golub *et al.* (1999) are among the first classical datasets used in bioinformatics.
- A selection of the dataset called `Golub_Merge` is contained in the `golubEsets` package, which is part of Bioconductor.
- The data consist of gene expression values for 7129 genes (rows) from 72 leukemia patients (columns).
  - 47 patients with acute lymphoblastic leukemia (ALL)
  - 25 patients with acute myeloid leukemia (AML)
  - The samples were assayed using Affymetrix Hgu6800 chips and data on the expression of 7129 genes (Affymetrix probes) are available.

Golub data

```
## Install bioconductor package 'golubEsets'
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("golubEsets")
library(golubEsets)
data(Golub_Merge)
```

```
## Golub data
?Golub_Merge
dim(Golub_Merge)
varLabels(Golub_Merge)
```

```
Golub_Merge$ALL.AML
table(Golub_Merge$ALL.AML)
Golub_Merge$Gender
```

```
## Expression data
dim(exprs(Golub_Merge))
head(exprs(Golub_Merge))
tail(exprs(Golub_Merge))
```

```
## Transpose the predictor x into n by p matrix
x <- t(as.matrix(exprs(Golub_Merge)))
y <- Golub_Merge$ALL.AML
dim(x)
sum(is.na(x))
```

```
g0 <- glmnet(x, y, alpha=0, family="binomial")
```

```
par(mfrow=c(1,2))
plot(g0, "lambda")
plot(g0, "norm")
```

```
g1 <- glmnet(x, y, alpha=1, family="binomial")
```

```
par(mfrow=c(1,2))
plot(g1, "lambda")
plot(g1, "norm")
```

```
set.seed(12345)
gvc <- cv.glmnet(x, y, alpha=1, family="binomial", nfolds=5)
gvc$lambda.min
gvc$lambda.1se
plot(gvc)
```

```
fit1 <- coef(gvc, s="lambda.min")
fit2 <- coef(gvc, s="lambda.1se")
c(sum(as.matrix(fit1)!=0), sum(as.matrix(fit2)!=0))
```

```
w1 <- which(as.matrix(fit1)!=0)
w2 <- which(as.matrix(fit2)!=0)
```

```
data.frame(name=colnames(x)[w1], beta=fit1[w1])
data.frame(name=colnames(x)[w2], beta=fit1[w2])
```

```
ge <- glmnet(x, y, alpha=0.5, family="binomial")
ge$df
ge$lambda
plot(ge, "lambda")
```

```
set.seed(111)
gecv <- cv.glmnet(x, y, alpha=0.5, family="binomial", nfolds=5)
plot(gecv)
```

```
fit3 <- coef(gecv, s="lambda.min")
fit4 <- coef(gecv, s="lambda.1se")
c(sum(as.matrix(fit3)!=0), sum(as.matrix(fit4)!=0))
```

```
w3 <- which(as.matrix(fit3)!=0)
w4 <- which(as.matrix(fit4)!=0)
data.frame(name=colnames(x)[w3], beta=fit3[w3])
data.frame(name=colnames(x)[w4], beta=fit4[w4])
```

```r
## Seperate training sets and test sets
Golub_Merge$Samples
tran <- Golub_Merge$Samples < 39
test <- !tran
c(sum(tran), sum(test))
```

```r
set.seed(123)
g1 <- cv.glmnet(x, y, alpha=1, family="binomial", K=5,
                subset=tran)
fit1 <- coef(g1, s="lambda.min")
fit2 <- coef(g1, s="lambda.1se")
c(sum(as.matrix(fit1)!=0), sum(as.matrix(fit2)!=0))
```

```r
p1 <- predict(g1, x[test,], s="lambda.min", type="class")
p2 <- predict(g1, x[test,], s="lambda.1se", type="class")
table(p1, y[test])
table(p2, y[test])
```

```
set.seed(1234)
g2 <- cv.glmnet(x, y, alpha=0.5, family="binomial", K=5,
                subset=tran)
fit3 <- coef(g2, s="lambda.min")
fit4 <- coef(g2, s="lambda.1se")
c(sum(as.matrix(fit3)!=0), sum(as.matrix(fit4)!=0))
```

```
p3 <- predict(g2, x[test,], s="lambda.min", type="class")
p4 <- predict(g2, x[test,], s="lambda.1se", type="class")
table(p3, y[test])
table(p4, y[test])
```

```
## 50 simulation replications
set.seed(111)
K <- 50
ERR <- DF <- array(0, c(K, 4))
for (i in 1:K) {
    tran <- as.logical(rep(0, 72))
    tran[sample(1:72, 38)] <- TRUE
    test <- !tran
    g1 <- cv.glmnet(x, y, alpha=1, family="binomial", K=5,
                    subset=tran)
    g2 <- cv.glmnet(x, y, alpha=0.5, family="binomial", K=5,
                    subset=tran)
    p1 <- predict(g1, x[test,], s="lambda.min", type="class")
    p2 <- predict(g1, x[test,], s="lambda.1se", type="class")
    p3 <- predict(g2, x[test,], s="lambda.min", type="class")
    p4 <- predict(g2, x[test,], s="lambda.1se", type="class")
    DF[i, 1] <- sum(coef(g1, s="lambda.min")!=0)
    DF[i, 2] <- sum(coef(g1, s="lambda.1se")!=0)
    DF[i, 3] <- sum(coef(g2, s="lambda.min")!=0)
    DF[i, 4] <- sum(coef(g2, s="lambda.1se")!=0)
    ERR[i, 1] <- sum(p1!=y[test])/sum(test)
    ERR[i, 2] <- sum(p2!=y[test])/sum(test)
    ERR[i, 3] <- sum(p3!=y[test])/sum(test)
    ERR[i, 4] <- sum(p4!=y[test])/sum(test)
}
```

```
apply(ERR, 2, mean)
DF
apply(DF, 2, var)
```