PQHS 471
Lecture 12: Housekeeping Utilities in Statistical Learning

Simulation: Random Number Generation, Permutation

Statistical Simulation

- Statistical simulation (Monte Carlo) is an important part of statistical method research.
- The statistical theories/methods are all based on assumptions. So most theorems state something like "if the data follow these models/assumptions, then...".
- The theories can hardly be verified in real world data because (1) the real data never satisfy the assumption; and (2) the underlying truth is unknown (no "gold standard").
- In simulation, data are "created" in a well controlled environment (model assumptions) and all truth are known. So the claim in the theorem can be verified.

Random Number Generator (RNG)

- Random number generator is the basis of statistical simulation. It serves to generate random numbers from predefined statistical distributions.
- Traditional methods (flip a coin or dice) work, but can't scale up.
- Computational methods are available to generate "pseudorandom" numbers.

Random Number Generator (RNG)

The random number generation often starts from generating uniform(0,1).
The most common method: "**Linear congruential generator**":

$$X_{n+1} = (aX_n + c) \text{ mod } m$$

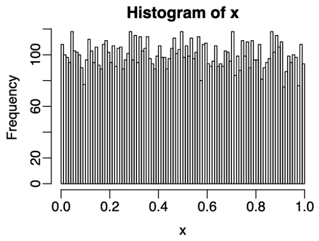
Here, a, c, and m are predefined numbers:

- X_0: random number "seed".
- a: multiplier, 1103515245 in *glibc*.
- c: increment, 12345 in *glibc*.
- m: modulus, 2^{32} or 2^{64}.

$U_n = X_n/m$ is distributed as Uniform(0,1).

Linear congruential generator

```
a = 1103515245; c = 12345; m = 2^32
n = 10000
x = numeric(n)
x[1] = 1
for( i in 2:n) {
x[i] = (a*x[i-1] + c) %% m
}
x = x/m
hist(x, 100)
```



Random Number Generator (RNG)

A few remarks about Linear congruential generator:

- The numbers generated will be exactly the same using the same seed.
- Want cycle of generator (number of steps before it begins repeating) to be large.
- Don't generate more than m/1000 numbers.

RNG in *R*:

- *set.seed* is the function to specify random seed.
- Read the help for *.Random.seed* for more description about random number generation in *R*.
- *runif* is used to generate uniform(0,1) r.v.

My recommendation: always set and save random number seed during simulation, so that the simulation results can be reproduced.

Simulate r.v. from other distributions

When the distribution has a **cumulative distribution function (cdf)** $\mathbf{F}$, the r.v. can be obtained by inverting the cdf ("inversion sampling"). This is based on the theory that the cdf is distributed as Uniform (0,1):
Algorithm: Assume $\mathbf{F}$ is the cdf of distribution $\mathcal{D}$. Given $u \sim unif(0,1)$, find a unique real number x such that $\mathbf{F}(x) = u$. Then $x \sim \mathcal{D}$.

Simulate r.v. from other distributions

When the distribution has a **cumulative distribution function (cdf) $\mathbf{F}$**, the r.v. can be obtained by inverting the cdf ("inversion sampling"). This is based on the theory that the cdf is distributed as Uniform (0,1):
Algorithm: Assume $\mathbf{F}$ is the cdf of distribution $\mathcal{D}$. Given $u \sim unif(0,1)$, find a unique real number x such that $\mathbf{F}(x) = u$. Then $x \sim \mathcal{D}$.
Example: exponential distribution. When $x \sim exp(\lambda)$, the cdf is $\mathbf{F}(x) = 1 - exp(-\lambda x)$. The inversion of cdf is:

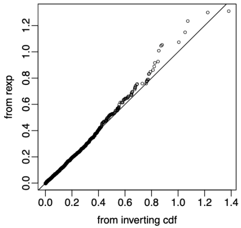
$$\mathbf{F}^{-1}(\mu) = -log(1-\mu)/\lambda$$

Then to generate exponential r.v., do:

- Generate uniform(0,1) r.v. denoted by μ.
- Calculate $x = -log(1-\mu)/\lambda$

Example: simulate exponential r.v.

```
lambda=5
u = runif(1000)
x = -log(1-u) / lambda
## generate from R's function
x2 = rexp(1000, lambda)
## compare
qqplot(x, x2, xlab="from inverting cdf", ylab="from rexp")
abline(0,1)
```



Simulate random vectors

Difficulty: Generating random vectors is more difficult, because we need to consider the correlation structure.
Solution: Generate **independent** r.v.'s, then apply some kind of transformation.
Example: simulate from multivariate normal distribution $MVN(\mu, \Sigma)$
Let Z be a p-vector of independent $N(0,1)$ r.v.'s, given $p \times p$ matrix D:

$$var(D^T Z) = D^T var(Z) D = D^T D$$

Therefore, the simulation steps are:

1. Perform **Cholesky decomposition** on Σ to find D s.t. $\Sigma = D^T D$.
2. Simulate $Z = (z_1, z_2, ..., z_p)' \sim N(0,1)$
3. Apply transformation $X = D^T Z + \mu$

R function mvrnorm available in *MASS* package.

Example: generate multivariate normal

```
## specify mean and variance/covariance matrix
mu = c(0,1)
Sigma = matrix(c(1.7, 0.5, 0.5, 0.8), nrow=2)
## Cholesky decomposition
D = chol(Sigma)
## generate 500 Z's.
Z = matrix(rnorm(1000), nrow=2)
## transform
X = t(D) %*% Z + mu
## check the means X
> rowMeans(X)
[1] -0.08976896  0.95802769
## check the variance/covariance matrix of X
> cov(t(X))
[,1]      [,2]
[1,] 1.7392114 0.5609027
[2,] 0.5609027 0.7380548
```

Permutation

- In statistical inference, it is important to know the distribution of some statistics under null hypothesis (H_0), so that quantities like p-values can be derived.
- The null distribution is available theoretically in some cases. For example,assume $X_i \sim N(\mu, \sigma^2), i = 1, ..., n$. Under $H_0 : \mu = 0$, we have $\bar{X} \sim N(0, \sigma^2/n)$. Then H_0 can be tested by comparing $\bar{X}$ with $N(0, \sigma^2/n)$.
- When null distribution cannot be obtained, it is useful to use **permutation test** to “create” a null distribution from data.

Permutation

The basic procedure of permutation test for H_0:

- Permute data under H_0 for a number of times. Each time recompute the test statistics. The test statistics obtained from the permuted data form the null distribution.
- Compare the observed test statistics with the null distribution to obtain statistical significance.

Permutation test example

Assume there are two sets of independent normal r.v.'s with the same known variance and unknown means: $X_i \sim N(\mu_1, \sigma^2), Y_i \sim N(\mu_2, \sigma^2)$.
We wish to test $H_0 : \mu_1 = \mu_2$.
Define the test statistics: $t = \bar{X} - \bar{Y}$. We know under the null, we have $t \sim N(0, 2\sigma^2/n)$ (assuming same sample size n in both groups). Using the permutation test, we do:

1. Pool X and Y together, denote the pooled vector by Z.
2. Randomly shuffle Z. For each shuffling, take the first n items as the new X (denote as X^*) and the next n items as the new Y (denoted as Y^*).
3. Compute $t^* = \bar{X^*} - \bar{Y^*}$.
4. Repeat steps 2 and 3 for a number of times. The result t^*'s form the null distribution of t.
5. To compute p-values, calculate $Pr(|t^*| > |t|)$.

Note: the random shuffling is based on H_0, that X and Y are i.i.d.

Example: permutation test

```
> x=rnorm(100, 0, 1)
> y=rnorm(100, 0.5, 1)
> t.test(x,y)
Welch Two Sample t-test
data:  x and y
t = -1.9751, df = 197.962, p-value = 0.04965

> nsims=50000
> t.obs = mean(x) - mean(y)
> t.perm = rep(0, nsims)
> for(i in 1:nsims) {
+     tmp = sample(c(x,y))
+     t.perm[i] = mean(tmp[1:100]) - mean(tmp[101:200])
+ }
> mean(abs(t.obs) < abs(t.perm))
[1] 0.04814
```

Permutation test: regression example

- Under linear regression setting (without intercept) $y_i = \beta x_i + \epsilon_i$. We want to test the coefficient: $H_0 : \beta = 0$.
- Observed data are (x_i, y_i) pairs.
- Use ordinary least square estimator for β, denote as $\hat{\beta}(\mathbf{x}, \mathbf{y})$.

The permutation test steps are:

1. Keep y_i unchanged, permute (change the order of) x_i to obtain a vector, denoted as x_i^*.
2. Obtain estimate under the permuted data: $\hat{\beta}^*(\mathbf{x}^*, \mathbf{y})$.
3. Repeat step 1 and 2. $\hat{\beta}^*$'s form the null distribution for $\hat{\beta}$.
4. P-value $= Pr(|\hat{\beta}^*| > \hat{\beta})$.

Note: the random shuffling of x_i is based on the H_0, that is there is no association between $\mathbf{x}$ and $\mathbf{y}$.

Example: regression permutation test

```
> x = rnorm(100); y = 0.2 * x + rnorm(100)
> summary(lm(y~x-1))
Coefficients:
Estimate Std. Error t value Pr(>|t|)
x    0.1502     0.1050    1.431    0.156
> nsims=5000
> beta.obs = coef(lm(y~x-1))
> beta.perm = rep(0, nsims)
> for(i in 1:nsims) {
+     xstar = sample(x)
+     beta.perm[i] = coef(lm(y~xstar-1))
+ }
> mean(abs(beta.obs) < abs(beta.perm))
[1] 0.157
```

Regularization

Linear Model Selection and Regularization

- Recall the linear model

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

- In the lectures that follow, we consider some approaches for extending the linear model framework.

In praise of linear models!

- 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 simple linear model can be improved, by replacing ordinary least squares fitting with some alternative fitting procedures.

Why consider alternatives to least squares?

- **Prediction Accuracy**: especially when $p > n$, to control the variance.
- **Model Interpretability**: By removing irrelevant features — that is, by setting the corresponding coefficient estimates to zero — we can obtain a model that is more easily interpreted. We will present some approaches for automatically performing **feature selection**.

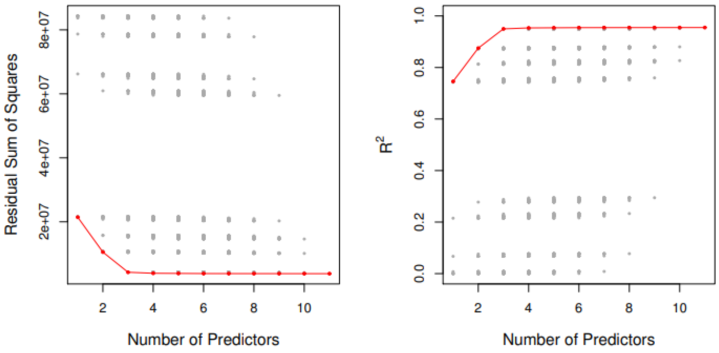
Three classes of methods

- **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 least squares 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 least squares 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 least squares.

Best Subset Selection

- 1. Let M_0 denote the **null model**, which contains no predictors. This model simply predicts the sample mean for each observation.
- 2. For $k = 1, 2, ...p$:
  - (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 M_k. Here **best** is defined as having the smallest RSS, or equivalently largest R^2.
- 3. Select a single best model from among $M_0, ..., M_p$ using cross-validated prediction error, C_p (AIC), BIC or adjusted R^2.

Example - Credit data set



For each possible model containing a subset of the ten predictors in the **Credit** data set, the RSS and R^2 are displayed. The red frontier tracks the **best** model for a given number of predictors, according to RSS and R^2 . Though the data set contains only ten predictors, the x-axis ranges from 1 to 11, since one of the variables is categorical and takes on three values, leading to the creation of two dummy variables.

Stepwise Selection

- For computational reasons, best subset selection cannot be applied with very large p. **Why not?**
- Best subset selection may also suffer from statistical problems when p is large: larger the search space, the higher the chance of finding models that look good on the training data, even though they might not have any predictive power on future data.
- Thus an enormous search space can lead to **overfitting** and high variance of the coefficient estimates.
- For both of these reasons, **stepwise** methods, which explore a far more restricted set of models, are attractive alternatives to best subset selection.

Forward Stepwise 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.

Forward Stepwise Selection: details

Forward Stepwise Selection

- 1. Let M_0 denote the **null** model, which contains no predictors.
- 2. For $k = 0, ...p - 1$:
  - (a) Consider all $p - k$ models that augment the predictors in M_k with one additional predictor.
  - (b) Choose the **best** among these $p - k$ models, and call it M_{k+1}. Here **best** is defined as having smallest RSS or highest R^2.
- 3. Select a single best model from among $M_0, ..., M_p$ using cross-validated prediction error, C_p (AIC), BIC or adjusted R^2.

More on Forward Stepwise Selection

- Computational advantage over best subset selection is clear.
- It is not guaranteed to find the best possible model out of all 2^p models containing subsets of the p predictors. **Why not? Give an example.**

Credit data example

# Variables	Best subset	Forward stepwise
One	rating	rating
Two	rating, income	rating, income
Three	rating, income, student	rating, income, student
Four	cards, income, student, limit	rating, income, student, limit

The first four selected models for best subset selection and forward stepwise selection on the **Credit** data set. The first three models are identical but the fourth models differ.

Backward Stepwise 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

Backward Stepwise Selection: details

Backward Stepwise Selection

- 1. Let M_p denote the **full** model, which contains p predictors.
- 2. For $k = p, p-1, ...1$:
    - (a) Consider all k models that contain all but one of the predictors in M_k, for a total of $k-1$ predictors.
    - (b) Choose the **best** among these k models, and call it M_{k-1}. Here **best** is defined as having smallest RSS or highest R^2.
- 3. Select a single best model from among $M_0, ..., M_p$ 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 $1 + \frac{p(p+1)}{2}$ 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.

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.

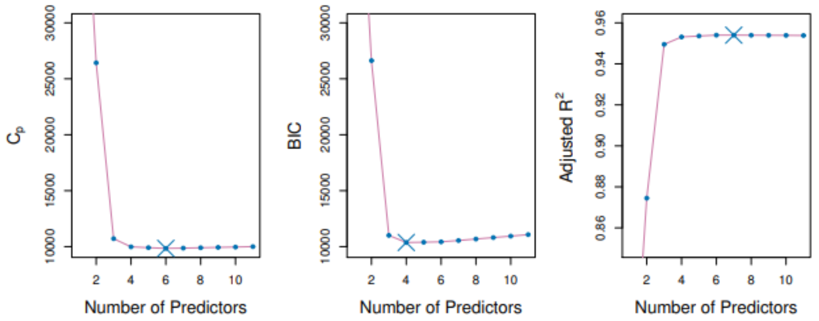
Estimating test error: two approaches

- We can **indirectly** estimate test error by making an **adjustment** to the training error to account for the bias due to overfitting.
- We can **directly** estimate the test error, using either a validation set approach or a cross-validation approach, as discussed in previous lectures.
- We illustrate both approaches next.

C_p, AIC, BIC, and Adjusted R^2

- These techniques adjust the training error for the model size, and can be used to select among a set of models with different numbers of variables.
- The next figure displays C_p, BIC, and adjusted R^2 for the best model of each size produced by best subset selection on the **Credit** data set.

Credit data example



Now for some details

- **Mallow's C_p**

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

  where d is the total $\#$ of parameters used and $\hat{\sigma}^2$ is an estimate of the variance of the error ϵ associated with each response measurement.

- The **AIC** criterion is defined for a large class of models fit by maximum likelihood:

$$AIC = -2 \log L + 2 \cdot d.$$

  where L is the maximized value of the likelihood function for the estimated model.

- In the case of the linear model with Gaussian errors, maximum likelihood and least squares are the same thing, and C_p and AIC are equivalent. **Prove this.**

Details on BIC

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

- 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 BIC 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. **See last figure**.

Adjusted R^2

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

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

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.
- Maximizing the adjusted R^2 is equivalent to minimizing $\frac{RSS}{n-d-1}$. While RSS always decreases as the number of variables in the model increases, $\frac{RSS}{n-d-1}$may increase or decrease, due to the presence of d in the denominator.
- Unlike the R^2 statistic, the adjusted R^2 statistic **pays a price** for the inclusion of unnecessary variables in the model. **See last figure**.

Validation and Cross-Validation

- Each of the procedures returns a sequence of models M_k indexed by model size $k = 0, 1, 2, ...$. Our job here is to select $\hat{k}$. Once selected, we will return model M_k.

Validation and Cross-Validation

- Each of the procedures returns a sequence of models 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 M_k.
- We compute the validation set error or the cross-validation error for each model 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 $\boldsymbol{\sigma^2}$** .
- 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 .

Shrinkage Methods

Ridge regression and **Lasso**

- 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.

Ridge regression

- Recall that the least squares fitting procedure estimates $\beta_0, \beta_1, ..., \beta_p$ using the values that minimize

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

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

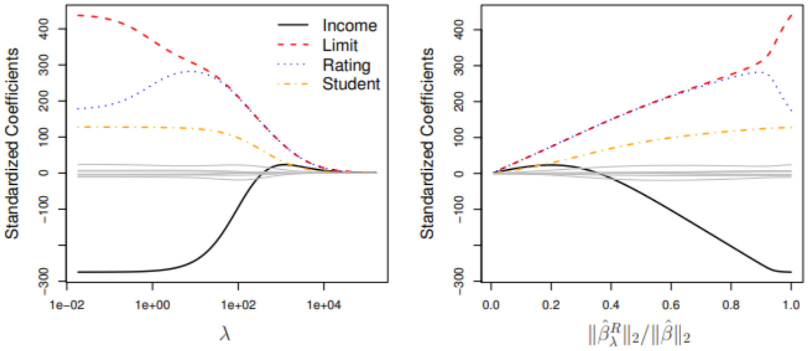
$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2$$

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

Ridge regression: continued

- 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 $\beta_1, ..., \beta_p$ 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.
- Selecting a good value for λ is critical; cross-validation is used for this.

Credit data example



Details of Previous Figure

- In the left-hand panel, each curve corresponds to the ridge regression coefficient estimate for one of the ten variables, plotted as a function of λ.
- The right-hand panel displays the same ridge coefficient estimates as the left-hand panel, but instead of displaying λ on the x-axis, we now display $||\hat{\beta}_\lambda^R||_2/||\hat{\beta}||_2$, where $\hat{\beta}$ denotes the vector of least squares coefficient estimates.
- The notation $||\beta||_2$ denotes the ℓ_2 norm (pronounced "ell 2") of a vector, and is defined as $||\hat{\beta}||_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$.

Ridge regression: 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

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

The 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, $\hat{\beta}_\lambda^L$, minimize the quantity

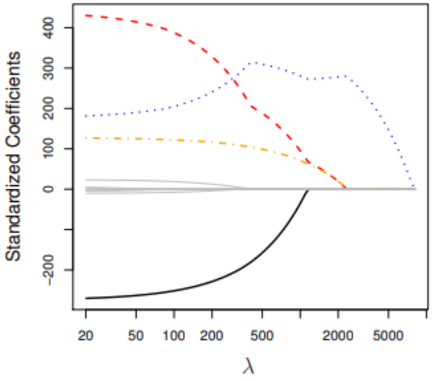
$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|$$

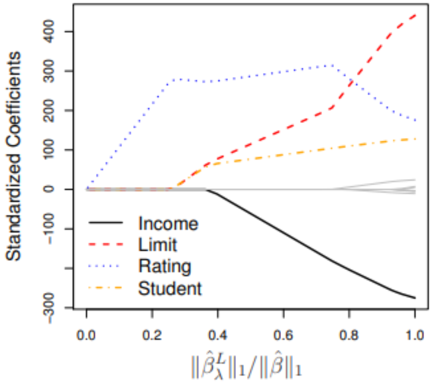
- In statistical parlance, the lasso uses an ℓ_1 (pronounced "ell 1") penalty instead of an ℓ_2 penalty. The ℓ_1 norm of a coefficient vector β is given by $||\beta||_1 = \sum|\beta_j|$.

The Lasso: continued

- As with ridge regression, the lasso shrinks the coefficient estimates towards zero.
- However, in the case of the 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.
- Hence, much like best subset selection, the lasso performs **variable selection**.
- We say that the lasso yields **sparse** models — that is, models that involve only a subset of the variables.
- As in ridge regression, selecting a good value of λ for the lasso is critical; cross-validation is again the method of choice.

Example: Credit dataset





The Variable Selection Property of the Lasso

Why is it that the 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

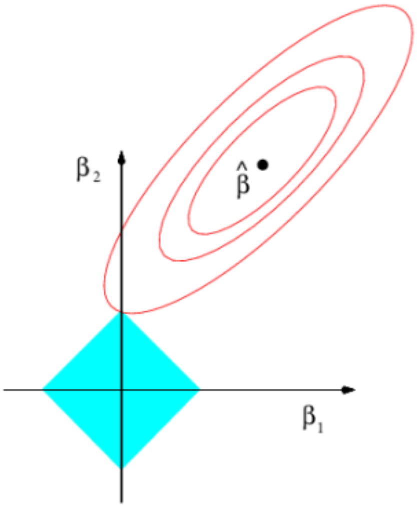
$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 \quad \text{subject to} \sum_{j=1}^{p}|\beta_j| \leq s$$

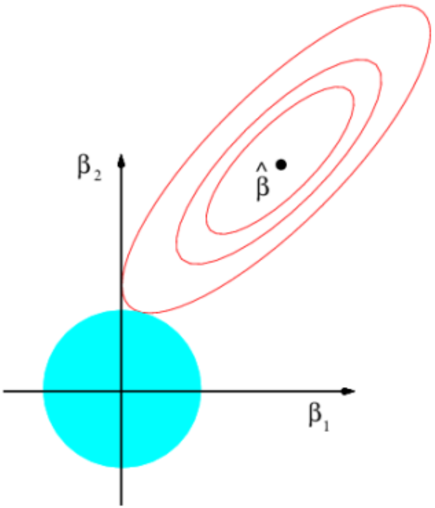
and

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

, respectively.

The Lasso Picture





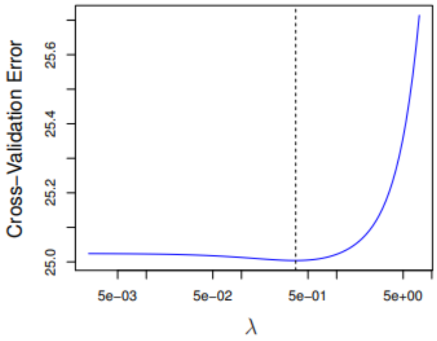
Conclusions

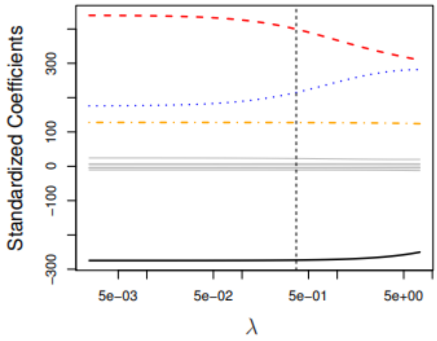
- These two examples illustrate that neither ridge regression nor the lasso will universally dominate the other.
- In general, one might expect the lasso to perform better when the response is a function of only a relatively small number of predictors.
- However, the number of predictors that is related to the response is never known **a priori** for real data sets.
- A technique such as cross-validation can be used in order to determine which approach is better on a particular data set.

Selecting the Tuning Parameter for Ridge Regression and Lasso

- 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.
- Finally, the model is re-fit using all of the available observations and the selected value of the tuning parameter.

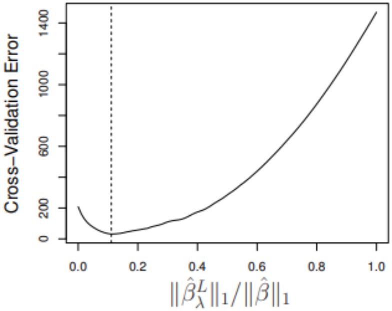
Credit data example

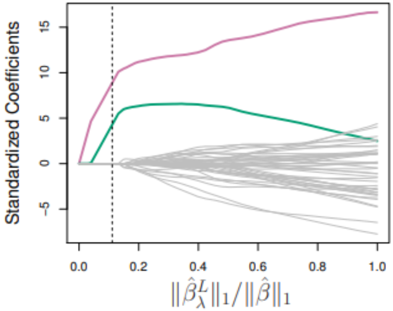




Left: Cross-validation errors that result from applying ridge regression to the **Credit** data set with various values of λ.
Right: The coefficient estimates as a function of λ. The vertical dashed lines indicates the value of λ selected by cross-validation.

Simulated data example





Left: Ten-fold cross-validation MSE for the lasso, applied to the sparse simulated data set from Slide 41.
Right: The corresponding lasso coefficient estimates are displayed. The vertical dashed lines indicate the lasso fit for which the cross-validation error is smallest.

Dimension Reduction Methods

- The methods that we have discussed so far in this chapter have involved fitting linear regression models, via least squares or a shrunken approach, using the original predictors, $X_1, X_2, ..., X_p$.
- We now explore a class of approaches that **transform** the predictors and then fit a least squares model using the transformed variables. We will refer to these techniques as **dimension reduction** methods.

Dimension Reduction Methods: details

- Let $Z_1, Z_2, ..., Z_M$ represent $M < p$ **linear combinations** of our original p predictors. That is,

$$Z_m = \sum_{j=1}^{p} \phi_{mj} X_j \tag{1}$$

  for some constants $\phi_{m1}, ..., \phi_{mp}$.

- We can then fit the linear regression model,

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \epsilon_i, \quad i = 1, ..., n, \tag{2}$$

  using ordinary least squares.

- Note that in model (2), the regression coefficients are given by $\theta_0, \theta_1, ..., \theta_M$. If the constants $\phi_{m1}, ..., \phi_{mp}$ are chosen wisely, then such dimension reduction approaches can often outperform OLS regression.

- Notice that from definition (1),

$$\sum_{m=1}^{M} \theta_m z_{im} = \sum_{m=1}^{M} \theta_m \sum_{j=1}^{p} \phi_{mj} x_{ij} = \sum_{j=1}^{p} \sum_{m=1}^{M} \theta_m \phi_{mj} xij = \sum_{j=1}^{p} \beta_j x_{ij},$$

  where

$$\beta_j = \sum_{m=1}^{M} \theta_m \phi_{mj} \tag{3}$$

- Hence model (2) can be thought of as a special case of the original linear regression model.
- Dimension reduction serves to constrain the estimated β_j coefficients, since now they must take the form (3).
- Can win in the bias-variance tradeoff.

Regularization summary

- Model selection methods are an essential tool for data analysis, especially for big datasets involving many predictors.
- Research into methods that give **sparsity**, such as the **lasso** is an especially hot area.