Statistical Machine Learning

Yiqiao YIN
Department of Statistics
Columbia University

Abstract

This document notes all materials discussed in Statistical Machine Learning, a course offered in Department of Statistics by Columbia University. We combine graduate level machine learning topics from Elements of Statistical Learning and R coding exercises from Introduction to Statistical Learning. This document also implements neural network and convolutional neural network from Stanford website. Most significantly, this document provides theoretical framework and evidence that better testing set accuracy can be achieved with the implementation of an interaction-based variable selection technique, I-score, The document sends a message to my audience that a better generation of statistical learning can be studied, a more accurate prediction methdology can be discovered, and a better future can be seen.

Contents

This document is dedicated to Professor Linxi Liu and Professor Shaw-Hwa Lo.

Artificial intelligence is a logical extension of human minds using machine power to execute human will.

— Yiqiao Yin

Preface

After a year as visting scholar at Columbia, I finally build up the foundation as well as my courage to take statistical machine learning. As the most important course to enter the realm of machine learning, it is essential to learn the theoretical framework behind approaches previous scholars have attempted.

The instructor of this course, Professor Linxi Liu, happened to be working with the same professor in Department of Statistics I have been working with. Through this common connection, a lot of interesting questions and sparks can be triggered deep into the field of machine learning and eventually the field can be pushed forward by my dedication.

In my point of view, artificial intelligence is a logical extension of our minds using machine power to execute human will. Serving as a foundation platform for artificial intelligence, the materials of this class too valuable to be ignored so I have decided to document everything I can about this topic from this class.

This document is structured in the following way. We start with basic topics such as parametric functions, loss functions, and bayesian models. Next, we move on to discuss PCA, LDA, quadratic fitting models for higher dimension techniques. This document will lands on Neural Network and Convolutional Neural Network, the most advanced machine learning methodology in the market right now.

I will then introduce I-score and Backward-dropping algorithm (developed by Professor Shaw-Hwa Lo in the Department of Statistics at Columbia University) as a variable selection methodology on sample datasets. This serves as a foundation document laying ground work of attempted research in the realm of machine learning. I will write a follow-up document with results that I-score is able to improve final testing set accuracy by identifying the variables that impact the responses the most.

To make this document practical for other users, the end of the document also introduces a various of techniques and I personally provided R code based on my experience or learning from other materials.

1 STATISTICAL LEARNING

1.1 Unsupervised and Supervised Learning

In unsupervised learning, we start with a matrix. We have quantitative measures such as weight, height, number of appearances, etc.. Our goal is to find 1) meaningful relationships between variables or units correlation analysis, 2) find low-dimensional representations of the data which make it easy to visualize the variables such as using PCA, ICA, multidimensional scaling, locally linear embeddings, etc., and/or 3) find meaningful clustering. Unsupervised learning is also known in statistics as exploratory data analysis.

In supervised learning, there are input variables, and output variables. If X is the vector of inputs for a particular sample. The output variable is variable is modeled by

$$Y = f(X) + \underbrace{\epsilon}_{\text{Random Error}}$$

The goal is simply to learn the function f, usinga set of training samples. The motivation is intuitive. We want to generate prediction. Prediction is useful when the input variable is readily available, but the output variable is not. We can also draw inferences. A model for f can help us understand the structure of teh data — which variables influence the output, and which does not? What is the relationship between between each variable and the output?

1.2 Parametric and Non-parametric

There are two kinds of supervised learning method: parametric methods and non-parametric methods. We assume that f takes a specific form. A linear form

$$f(X) = X_1\beta_1 + \cdots + X_p\beta_p \text{ while } Y \sim \mathcal{N}(\sum_{j=1}^{p}\beta_j x_j, \sigma^2); \epsilon \sim \mathcal{N}(0, \sigma^2)$$

with parameters $\beta_1, ..., \beta_p$. Using the training data, we try to fit the parameters. For non-pamaetric method, we do not make any assumptions on the form of f, but we restrict how "wiggle" or "rough" the function can be.

1.3 Loss Function

The loss function $L(Y, \hat{f}(X))$ measures the errors between the observed value Y and the predicted value $\hat{f}(X)$. In a regression problem, two most common loss functions are

$$L(Y, \hat{f}(X)) = \begin{cases} (Y - \hat{f}(X))^2, & \text{squared error} \\ |Y - \hat{f}(X), & \text{absolute error} \end{cases}$$

The prediction error is given based on the following. Given training data: $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, the predicted function is $\hat{f}$. The goal in supervised learning is to minimzie the expected prediction error. Under squared-error loss, this is the Mean Squared Error.

$$MSE(\hat{f}) = E(y_0 - \hat{f}(x_0))^2.$$

Unfortunately, this quantity cannot be compputed, because we do not know the joint distribution of (X, Y). We can compute a sample average using the training data; this is known as the training MSE:

$$MSE_{\text{training}}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2.$$

The main challenge of statstical learning is that a low training MSE does not imply a low MSE. If we have test data $\{(x_i', y_i'); i = 1, ..., m\}$ which were not used to fit the model, a better measure of quality for $\hat{f}$ is the test MSE:

$$MSE_{\text{test}}(\hat{f}) = \frac{1}{m}\sum_{i=1}^{m}(y_i' - \hat{f}(x_i'))^2.$$

1.3.1 Bias Variance Decomposition

Let x_0 be a fixed test point, $y_0 = f(x_0) + \epsilon_0$, and $\hat{f}$ be estimated from n training samples $(x_1, y_1), ..., (x_n, y_n)$. Let E denote the expectation over y_0 and the training outputs $(y_1, ..., y_n)$. Then, the MSE at x_0 can be decomposed

$$MSE(x_0) = E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + Var(\epsilon)_0).$$

Observe the last term, $Var(\epsilon)_0$ is irreducible error. The variance of the estimates of Y is $E[\hat{f}(x_0) - E(\hat{f}(x_0))]^2$. This measures how much the estimate of $\hat{f}$ at x_0 changes when we sample new trainig data.

The above equation tells us that in order to minimize the expected test error, we need to select a statistical learning method that simultaneously achieves low variance and low bias. Note that the variance is inherently a nonnegative quantity, and squared bias is also nonnegative. Hence, we see that the expected test MSE can never lie below Var(ϵ), the irreducible error from the the formula.

In details, variance refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. Since the training data are used to fit the statistical learning method, different training data sets will result in a different $\hat{f}$. But ideally the estimate for f should not vary too much between training sets. However, if a method has high variance then small changes in the training data can result in large changes in $\hat{f}$. On the other hand, bias refers to the error that is introduced by approximating a real-life problem, which may be complicated.

2 CLASSIFICATION PROBLEM

In classification setting, the output takes values in a discrete set. For example, if we are predicting the brand of a car based on a number of variables, the function f takes values in the set such as $\{\text{Ford}, \text{Toyota}, ...\}$. In this case, The model $Y = f(X) + \epsilon$ becomes insufficient, as f is not necessarily real-valued. We will use slightly different notation. $P(X,Y)$, the joint distribution of (X,Y), $P(Y|X)$, the conditional distribution of X given Y, and $\hat{y}_i$, the prediction for x_i. In this case a common 0-1 loss would be

$$\mathbb{E}(\mathbf{1}(y_0 \neq \hat{y_0}))$$

2.1 Bayesian Model

2.1.1 Maximum Likelihood Estimation

We have the following setup. Given data: $x_1, ..., x_n$, parametric model $\mathcal{P} = \{p(x|\theta)|\theta \in \mathcal{T}\}$, the objective is to find the distribution in $\mathcal{P}$ which best explains the data. That means we have to choose a "best" parameter value $\hat{\theta}$.

Maximum Likelihood assumes that the data is best explained by the distribution in $\mathcal{P}$ under which it has the highest probability (or the highest density value). Hence, the **maximum likelihood estimator** is defined as

$$\hat{\theta}_{\text{ML}} := \arg\max_{\theta \in \mathcal{T}} p(x_1, ..., x_n|\theta)$$

the parameter which maximizes the joint density of the data.

Here we need to make a crucial assumption, i.e., the iid. assumption. The standard assumption of ML methods is that the data is independent and identically distributed, i.i.d., that is, generated by independently sampling repeatedly from the same distribution P. If the density of P is $p(x|\theta)$, that means the joint density decomposes as

$$(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i|\theta)$$

The analytic criterion for a maximum likelihood estimator (under the i.i.d. assumption) is

$$\nabla_\theta \left(\prod_{i=1}^{n} p(x_i|\theta) \right) = 0$$

We use the "logarithm trick" to avoid a huge product rule computation. That is,

$$\begin{aligned} \hat{\theta}_{\text{ML}} &= \arg\max_{\theta} \textstyle\prod_{i=1}^{n} p(x_i|\theta) \\ &= \arg\max_{\theta} \log \left(\textstyle\prod_{i=1}^{n} p(x_i|\theta) \right) \\ &= \arg\max_{\theta} \textstyle\sum_{i=1}^{n} \log p(x_i|\theta) \end{aligned}$$

Analytic maximality criterion would be

$$0 = \sum_{i=1}^{n} \nabla_\theta \log p(x_i|\theta) = \sum_{i=1}^{n} \frac{\nabla_\theta p(x_i|\theta)}{p(x_i|\theta)}$$

and we do not always have a solution depending on what model we use.

2.1.1.1 Ex: Gaussian Mean MLE

Consider Gaussian density in one dimension

$$g(x;\mu,\sigma) := \frac{1}{\sqrt{2\pi\sigma}} \exp\big(-\frac{(x-\mu)^2}{2\sigma^2}\big)$$

The quotient $\frac{x-\mu}{\sigma}$ measures deviation of x from its expected value in units of σ (i.e. σ defines the length scale). The d dimensions Gaussian density, we have the quadratic function

$$-\frac{(x-\mu)^2}{2\sigma^2} = -\frac{1}{2}(x-\mu)(\sigma^2)^{-1}(x-\mu)$$

is replaced by a quadratic form:

$$g(x;\mu,\Sigma) := \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\big(-\frac{1}{2} < (x-\mu), \Sigma^{-1}(x-\mu) > \big)$$

For multivariate Gaussians, the model $\mathcal{P}$ is the set of all Gaussian densities on $\mathbb{R}^d$ with fixed covariance matrix Σ,

$$\mathcal{P} = \{g(\cdot|\mu,\Sigma)|\mu \in \mathbb{R}^d\}$$

where g is the Gaussian density function. The parameter space is $\mathcal{T} = \mathbb{R}^d$. For the MLE, we solve the following:

$$\begin{array}{rcl}
\text{Solve } \sum_{i=1}^n \nabla_\mu \log g(x_i|\mu,\Sigma) & = & 0 \\
\text{Setup: } 0 & = & \sum_{i=1}^n \nabla_\mu \log \frac{1}{\sqrt{(2\pi)^d|\Sigma|}} \exp\big(-\frac{1}{2} < (x_i-\mu), \Sigma^{-1}(x_i-\mu) > \big) \\
 & = & \sum_{i=1}^n \nabla_\mu \log \Bigg(\frac{1}{\sqrt{(2\pi)^d|\Sigma|}} \exp\Bigg) + \log \Bigg(\big(-\frac{1}{2} < (x_i-\mu), \Sigma^{-1}(x_i-\mu) > \big)\Bigg) \\
 & = & \sum_{i=1}^n \nabla_\mu \big(-\frac{1}{2} < (x_i-\mu), \Sigma^{-1}(x_i-\mu) > \big) \\
 & = & -\sum_{i=1}^n \Sigma^{-1}(x_i-\mu) \\
\text{Solve for} & & \\
0 & = & \sum_{i=1}^n (x_i-\mu) \\
\Rightarrow \mu & = & \frac{1}{n}\sum_{i=1}^n x_i
\end{array}$$

We can conclude that the maximum likelihood estimator of the Gaussian expectation parameter for fixed covariance is

$$\hat{\mu}_{\text{ML}} := \frac{1}{n}\sum_{i=1}^n x_i$$

2.1.2 Bayes' Theorem

The defining assumption of Bayesian statistics is that the distribution P_θ which models the data is a random quantity and itself has a distribution A. The generative model for data $X_1, X_2, ...$ is

$$P_\theta \sim Q$$
$$X_1, X_2, ... \overset{i.i.d.}{\sim} P_\theta$$

The rational behind the approach is that 1) in any statistical approach (Bayesian or frequentist), the distribution P_θ is unknown, 2) Bayesian statistics argues that any form of uncertainty should be expressed by probability distributions. 3) We can think of the randomness in Q as a model of the statistician's lack of knowledge regarding P_θ. The distribution Q is called the priori distribution of the prior. We use q to denote its density if it exists. Our objective is to determine the conditional probability of P given observed data

$$\Pr(\theta | x_1, ..., x_n).$$

The distribution is called the posteriori distribution or posterior. Given data, $X_1, ..., X_n$, we can compute the posterior by

$$\Pr(\theta | x_1, ..., x_n) = \frac{(\prod_{i=1}^n p(x_i|\theta))q(\theta)}{p(x_1, ..., x_n)} = \frac{(\prod_{i=1}^n p(x_i|\theta))q(\theta)}{\int (\prod_{i=1}^n p(x_i|\theta))q(\theta)}$$

The individual terms have names

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

2.1.3 MAP Estimation

Suppose $\prod(\theta | x_{1,n})$ is the posterior of a Bayesian model. The estimator

$$\hat{\theta}_{\text{MAP}} = \arg\max_\theta \prod(\theta | x_{1,n})$$

is called the maximum a posteriori (or MAP) estimator for θ.

For linear mapping, we define the following. A matrix $X \in \mathbb{R}^{n \times m}$ defines a lienar mapping $f_x : \mathbb{R}^m \to \mathbb{R}^n$. The image of a mapping f is the set of all possible function values, here

$$\text{image}(f_X) := \{y \in \mathbb{R}^n | Xz = y \text{ for some } z \in \mathbb{R}^m\}$$

The image of a linear mapping $\mathbb{R}^m \to \mathbb{R}^n$ is a linear subspace of $\mathbb{R}^n$. The columns of X form a basis of the image space:

$$\text{image}(\bar{X}) = \text{span}\{X_1^{\text{col}}, ..., X_m^{\text{col}}\}$$

This is one of the most useful things for matrices and we can interpret as a linear combination of columsn form the target image.

2.1.4 Symmetric and Orthogonal Matrices

Given the cocnepts of linear mapping, the theorems from real analysis follow as well. We can introduce column ranks, invertibility, one-one, and etc.. For orthogonal matrices, we have the following definition: A matrix $\mathcal{O} \in \mathbb{R}^{m\times m}$ is called orthogonal is $\mathcal{O}^{-1} = \mathcal{O}^T$. Orthogonal matrices describe two types of operations: 1) rotations of the coordinate system, and 2) permutations of the coordinate axes. Symmetric matrices are defined as the follows. A matrix $A \in \mathbb{R}^{m\times m}$ is called symmetric if $A = A^T$. Note that symmetric and orthogonal matrices are very different objects.

Based on the definitions above, we raise the concept of orthonormal basis, ONB. A basis $\{v_1, ..., v_m n\}$ of $\mathbb{R}^m$ is called an orthonormal basis if

$$< v_i, v_j >= \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

In other words, the v_i are pairwise othogonal and each of them with length 1. A matrix is orthogonal precisely if its rows from an ONB. Any two ONBs can be transformed into each other by an orthogonal matrix.

To represent a basis, suppose $\mathcal{E} = \{e_1, ..., e_d\}$ is a basis of a vector space. Then a vector x is represented as $x = \sum_{j=1}^d [x_j]_{\mathcal{E}} e^{(j)}$ while $[x_j]_{\mathcal{E}} \in \mathbb{R}$ are the coordinates of x w.r.t. $\mathcal{E}$. We can have other bases as well. Consider $\mathcal{B} = \{b_1, ..., b_d\}$ is another basis. Then x can be represented alternative as $x = \sum_{j=1}^d [x_j]_{\mathcal{B}} b^{(j)}$.

Change-of-basis matrix: The matrix $M := \left([e^{(1)}]_{\mathcal{B}}, ..., [e^{(d)}]_{\mathcal{B}}\right)$. If both $\mathcal{E}$ and $\mathcal{B}$ are ONBs, M is orthogonal.

The matrix representing a linear mapping $A : \mathbb{R}^d \to \mathbb{R}^d$ in the basis $\mathcal{E}$ is computed as

$$[A]_{\mathcal{E}} := \left([A(e^{(1)})_{\mathcal{E}}, ..., [A(e^{(d)})_{\mathcal{E}}\right)$$

The matrix representing a linear mapping also changes when we change basis $[A]_{\mathcal{B}} = M[A]_{\mathcal{E}}M^{-1}$. Applied to a vector x, this means

$$[A]_{\mathcal{B}}[x]_{\mathcal{B}} = \underbrace{M}_{\text{Transform x back to } \mathcal{B}} \overbrace{[A]_{\mathcal{E}}}^{\text{Apply } A \text{ in representation } \mathcal{E}} \underbrace{M^{-1}}_{\text{transform } x \text{ back to } \mathcal{B}} [x]_{\mathcal{B}}$$

2.2 EM Algorithm

In statistics, an expectation-maximization (EM) algorithm is an iterative method to find maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.

2.2.1 Jensen's Inequality

Let f be a function whose domain is the set of real numbers. Recall that f is a convex function if $f''(x) \geq 0$ for all $x \in \mathbb{R}$. In ths case of f taking vector-valued inputs, this is generalized to the condition that its hessian H is positive semi-definite ($H \geq 0$). If $f''(x) > 0$ for all x, then we say f is strictly convex (in the vector-valued case, the corresponding statement is that H must be positive definite, written $H > 0$). Hensen's inequality can then be stated as follows:

Theorem. Let f be a convex function, and let X be a random variable. Then we have

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}(X))$$

Moreover, if f is strictly convex, then $\mathbb{E}[f(X)] = f(\mathbb{E}(X))$ holds true if and only if $X = \mathbb{E}[X]$ with probability 1 (i.e., if X is a constant). ### The EM Algorithm

Suppose we have an estimation problem in which we have a training set $\{x^{(1)}, ..., x^{(m)}\}$ consisting of m independent examples. We wish to fit the parameters of a model $p(x, z)$ to the data, where the likelihood is given by

$$\begin{array}{rcl} l(\theta) & = & \sum_{i=1}^{m} \log p(x;\theta) \\ & = & \sum_{i=1}^{m} \log \sum_{i} p(x, z; \theta). \end{array}$$

But, explicitly finding the MLE of the parameters θ may be hard. In this case, the $z^{(i)}$'s are the latent random variables; and it is often the case that if the $z(i)$'s were observed, then maximum likelihood estimation would be easy.

In such a setting, the EM algorithm gives an efficient method for maximum likelihood estimation. Maximizing $l(\theta)$ explicitly might be difficult, and our strategy will be to instead repeatedly construct a lower-bound on l (E-step), and then optimize that lower-bound (M-step).

For each i, let Q_i be some distribution over the z's ($\sum_z Q_i(z) = 1$, $Q_i(z) \geq 0$). Consider the following

$$\begin{array}{rcl} \sum_i \log p(x^{(i)};\theta) & = & \sum_i \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta) \\ & = & \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \\ & \geq & \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \end{array}$$

The last step of this derivation used Jensen's inequality. Specifically, $f(x) = \log x$ is a concave function, since $f''(x) = -1/x^2 < 0$ over its domain $x \in \mathbb{R}^+$. Moreover, the term

$$\sum_{z^{(i)}} Q_i(z^{(i)}) \left[\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

in the summation is just an expectation of the quantity $[p(x^{(i)}), z^{(i)}; \theta)/Q_i(z^{(i)})]$ with respect to $z^{(i)}$ drawn according to the distribution given by Q_i. By Jensen's inequality, we have

$$f\left(E_{z^{(i)} \sim Q_i} \left[\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]\right) \geq E_{z^{(i)} \sim Q_i} f\left[\left(\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right) \right]$$

where the $z^{(i)} \sim Q_i$ subscripts above indicate that the expectations are with respect to $z^{(i)}$ drawn from Q_i.

We need to make the lower-bound tight at the value of θ. To make this happen, we need for the step involving Jensen's inequality in our derivation above to hold with equality. We require that

$$\frac{p(x^{(i)}, z^{(I)}; \theta)}{Q_i(z^{(i)})} = c$$

for some constant c that does not depend on $z^{(i)}$. This is easily accomplished by shooing

$$Q_i(z^{(i)}) \propto p(x^{(i)}, z^{(i)}; \theta)$$

Actually, since we know that $\sum_z Q_i(z^{(i)}) = 1$, this further tells us that

$$\begin{aligned} Q_i(z^{(i)}) &= \tfrac{p(x^{(i)},z^{(i)};\theta)}{\sum_z p(x^{(i)},z;\theta)} \\ &= \tfrac{p(x^{(i)},z^{(i)};\theta)}{p(x^{(i)};\theta)} \\ &= p(z^{(i)}|x^{(i)};\theta) \end{aligned}$$

Thus, we simply set the Q_i's to be the posterior distribution of the $z^{(i)}$'s given $x^{(i)}$ and setting of the parameters θ.

Now we want to maximize the lower-bound on loglikelihood l. This is the E-step. In the M-step of the algorithm, we maximize our formula $\sum_i \log p(x^{(i)};\theta) \geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}$ with respect to the parameters to obtain a new setting of the θ's. Repeatedly carry out these two steps gives us the EM algorithm, which is the following

Repeat until convergence

(E-Step) For each i, set

$$Q_i(z^{(i)}) := p(z^{(i)}|x^{(i)};\theta).$$

(M-Step) Set

$$\theta := \arg\max_\theta \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}.$$

2.2.2 Will it Converge?

Suppose $\theta^{(i)}$ and $\theta^{(i+1)}$ are the parameters from two successive iterations of EM. We will now prove that $l(\theta^{(i)}) \leq l(\theta^{(i+1)})$, which shows EM always monotonically improves the log-likelihood. The key to showing this result lies in our choice of the Q_i's. Specifically, on the iteration of EM in which the parameters had started out as $\theta^{(i)}$, we would have chosen $Q_i^{(i)}(z^{(i)}) := p(z^{(i)}|x^{(i)};\theta^{(t)})$. Applying Jensen's inequality to formula $\sum_i \log p(x^{(i)};\theta) \geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}$, holds with equality, and hence

$$l(\theta^{(i)}) = \sum_i \sum_{z^{(i)}} Q_i^{(i)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(i)})}{Q_i^{(i)}(z^{(i)})}.$$

The parameters $\theta^{(t+1)}$ are then obtained by maximizing the right hand side of the equation above. Thus,

$$\begin{aligned} l(\theta^{(t+1)}) &\geq \textstyle\sum_i \sum_{z^{(i)} Q_i^{(i)}} Q_i^{(i)}(z^{(i)}) \log \tfrac{p(x^{(i)},z^{(i)};\theta^{(t+1)})}{Q_i^{(t)}(z^{(i)})} \\ &\geq \textstyle\sum_i \sum_{z^{(i)}} Q_i^{(i)}(z^{(i)}) \log \tfrac{p(x^{(i)},z^{(i)};\theta^{(i)})}{Q_i^{(i)}(z^{(i)})} \\ &= l(\theta^{(i)}) \end{aligned}$$

The first inequality comes from the fact that

$$l(\theta) \geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

holds for any values of Q_i and θ, and in particular holds for $Q_i = Q_i^{(i)}$, $\theta = \theta^{(i+1)}$. To get the second to last equation in the derivation above, we used the fact that $\theta^{(t+1)}$ is chosen explicitly to be

$$\arg\max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})},$$

and thus this formula evaluated at $\theta^{(t+1)}$ must be equal to or larger than the same formula evaluted at $\theta^{(i)}$. Finally, we get to the last equation in the derivation and follows from $q_i^{(i)}$ having been chosen to make Jensen's inequality hold with equality at $\theta^{(i)}$.

2.3 Logistic

To model the relationship between $p(X) = Pr(Y = 1|X)$ and X, logistic function is a good candidate. In logistic regression, the function takes the following form

$$p(X) = \frac{\exp \beta_0 + \beta_1 X}{1 + \exp \beta_0 + \beta_1 X}$$

To fit this model, we use a method called maximum likelihood. We rewrite $p(X)$ into

$$\frac{p(X)}{1 - p(X)} = \exp \beta_0 + \beta_1 X$$

Taking logarithm on both sides, we arrive

$$\log \left(\frac{p(X)}{1 - p(X)} \right) = \beta_0 + \beta_1 X$$

The left-hand side, which is called logit, is the response we want estimate. Statistical inference is needed to compute the estimated regression coefficients. The coefficients β_0 and β_1 in the definition are unknown, and must be estimated based on the available training data. A general method is to maximum likelihood since it has better statistical properties. The intuition behind using maximum likelihood to fit a logistic regression model is as follows: we seek estimates for β_0 and β_1 such that the predicted probability $\hat{p}(x_i)$ of default for each individual corresponds as closely as possible to the individual's observed default status. This intuition can be formalized using a likelihood function:

$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'}))$$

The estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are chosen to maximize this likelihood function. Predictions can be made once the coefficients have been estimated,

$$\hat{p}(X) = \frac{\exp(\hat{\beta}_0 + \hat{\beta}_1 X)}{1 + \exp(\hat{\beta}_0 + \hat{\beta}_1 X)}.$$

For multiple logistic regression, the model takes the following generalized form

$$\log \left(\frac{p(X)}{1 - p(X)} \right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,$$

where $X = (X_1, ..., X_p)$ are p predictors. The left-hand side is called the log-odds or logit.

2.4 Linear Discriminant Analysis (LDA)

For the following situations, one might want to consider the validty of logistic regression and pursue linear discriminant analysis. When the classes are well-separated, the parameter estiamtes for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem. If n is small and the distribution of the predictors X is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.

Suppose thereis $p = 1$ one predictor and we would like to estimate $f_k(x)$ to estimate $p_k(x)$. Suppose we assume $f_k(x)$ is normal or Gaussian. The normal density takes the form

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(- \frac{1}{2\sigma_k^2}(x - \mu_k)^2 \right)$$

where μ_k and σ_k^2 are the mean and variance parameters for the kth class. Plugging the normal density formula into Bayes' Theorem, we get

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2}(x - \mu_k)^2)}{\sum_{l=1}^K \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2}(x - \mu_l)^2)}$$

while π_k denotes the prior probability that an observation belongs to the kth class. Taking the log of and rearranging the terms, it is not hard to show that

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k}{2\sigma^2} + \log(\pi_k)$$

is the largest. For example, consider $K = 2$ and $\pi_1 = \pi_2$. The Bayes classifier assigns an observation to class 1 if $2x(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2$, and to class 2 otherwise. In this case, we have Bayes' decision boundary

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2}.$$

In practice, we still need to estimate the parameters $\mu_1, ..., \mu_K$, $\pi_1, ..., \pi_K$ and σ^2 even though we are quite certain that our observation is drawn from a Gaussian distribution. The linear discriminant analysis (LDA) method approximates the Bayes classifier by using estsiamtes for π_k, μ_k, and σ^2. In particular, the estiamtes are

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^K \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$$

where n is the total number of training observations, and n_k is the number of training observations in the kth class while $\hat{\sigma}^2$ can be seen as a weighted average of the sample variances for each of the K classes.

LDA estimates π_k using the proportion of the training observations that belong to the kth class. In other words,

$$\hat{\pi}_k = n_k/n.$$

The LDA classifiers use the above estiamtes and assign an observation $X = x$ to the class for which

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma^2}} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k)$$

is the largest. The word linear in the classifier's name stems from the fact the discriminant functions $\hat{\delta}_k(x)$ are linear functions of x.

3 UNSUPERVISED LEARNING

3.1 Principal Component Analysis (PCA)

As the most popular unsupervised procedure, invented by Karl Pearson (1901), and developed by Harold Hotelling (1993), principal component analysis provides a way to visualize high dimensional data, summarizing the most important information. Let X be a data matrix with n samplesl, and p variables. From each variable, we subtract the mean of the column; i.e. we center the variables.

Principal Component Analysis assumes the following. The directions along which uncertainty in data is maximal are most interesting. The uncertainty is measured by variance. The algorithm takes the following steps. Consider a data set with D dimensions:

1) Compute empirical covariance matrix of the data;
2) Compute its eigen-values $\lambda_1, ..., \lambda_D$, and eigen-vectors $\xi_1, ..., \xi_D$;
3) Choose the d largest eigen-values, say, $\lambda_{j1}, ..., \lambda_{jd}$;
4) Define subspace as $V := \text{span}\{\xi_{j1}, ..., \xi_{jd}\}$;
5) Project data onto V: for each x_i, compute $x_i^v := \sum_{j=1}^d < x_i, \xi_j > \xi_j$.

Several notation here takes the following form. Empirical mean of the data is $\hat{\mu}_n := \frac{1}{n}\sum_{i=1} n x_i$. Empirical variance of data (1 dimension) is $\hat{\sigma}_n^2 := \frac{1}{n}\sum_{i=1}^n (x_i - \hat{\mu}_n)^2$. Empirical covariance of data (D dimensions) is $\hat{\Sigma}_n := \frac{1}{n}\sum_{i=1}^n (x_i - \hat{\mu}_n)(x_i - \hat{\mu}_n)^t$.

The algorithm aims to project data onto a direction $v \in \mathbb{R}^D$ such that the variance of the projected data is maximized.

The first principal component of a set of features $X_1, X_2, ..., X_p$ is the normalized linear combination of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$$

that has the largest variance. By normalized, we mean that $\sum_{j=1}^p \phi_{j1}^2 = 1$. We refer to the elements $\phi_{11}, ..., \phi_{p1}$ as the loadings of the first principal component; together, the loadings make up the principal component loading vector, $\phi_1 = (\phi_{11}\phi_{21}\dots\phi_{p1})^T$. We constrain the loadings so that their sum of squares is equal to one.

To find the first principal component $\phi_1 = (\phi_{11}, ..., \phi_{p1})$, we solve the following optimization

$$\max_{\phi_{11},...,\phi_{p1}} \left\{ \frac{1}{n}\sum_{i=1}^n \left(\sum_{j=1}^p \phi_{j1}x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^p \phi_{j1}^2 = 1.$$

Projection of the ith sample onto ϕ_1 is also known as the score z_{i1}. The variance of the n samples is also projected onto ϕ_1. To find the second principal component $\phi_2(\phi_{12}, ..., \phi_{p2})$, we solve the folowing optimization

$$\max_{\phi_{12},...,\phi_{p2}} \left\{ \frac{1}{n}\sum_{i=1}^n \left(\sum_{j=1}^p \phi_{j2}x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^p \phi_{j2}^2 = 1 \text{ and } \sum_{j=1}^p \phi_{j1}\phi_{j2} = 0.$$

The first and second principal components must be orthogonal, which is equivalent as saying that the scores $(z_{11}, ..., z_{n1})$ and $(z_{12}, ..., z_{n2})$ are uncorrelated. The optimization is fundamental in linear algebra. It is satisfied by either the singular value decomposition (SVD) or X: $X = U\Sigma\Phi^T$, where the ith column of Φ is the ith principal copmonent ϕ_i, and the ith column of $U\Sigma$ is the ith vector of scores $(z_{1i}, ..., z_{ni})$. The eigendecomposition of X^TX: $X^TX = \Phi\Sigma^2\Phi^T$.

3.1.1 Mathematis of Principal Components

we start with p-dimensional vectors, and want to summarize them by projecting down into a q-dimensional subspace. The summary will be the projection of the original vectors on to q directions, the principal components, which span the subspace. There are several equivalent ways of deriving the principal components mathematically. The simplest one is by finding the projections which maximize the variance. The first principal component is the direction in space along which projections have the largest variance. The second principal component is hte direction which maximizes variance among all directions orthogonal to the first. The kth component is the variance-maximizing direction orthogonal to the previous $k-1$ components. There are p principal components in all.

Rather than maximizing variance, it might sound more plausible to look for the projection with the smallest average (mean-squared) distance between the original vectors and their projections on to the principal components; this turns out to be equivalent to maximizing the variance.

3.1.2 Minimizing Projection Residuals

Consider a p-dimensional vector and we want to proejct them on to a line through the origin. We can specify the line by a unit vector along it, w, and then the projection of a data vector x_i on to the line that is $x_i \cdot w$ which is a scalar. This is the distance of the projection from the origin; the actual coordinate in p-dimensional space is $(x_i \cdot w)w$. The mean of the projections will be zero, because the mean of the vectors x_i is zero:

$$\frac{1}{n}\sum_{i=1}^{n}(x_i \cdot w)w = \left(\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) \cdot w\right)w$$

If we try to use our projected or image vectors instead of our original vectors, there will be some error, because (in general) the images do not coincide or residual of the projection. How big is it?

$$\begin{aligned} ||x_i - (w \cdot x_i)w||^2 &= (x_i - (w \cdot x_i)w) \cdot (x_i - (w \cdot x_i)w) \\ &= x_i \cdot x_i - x_i \cdot (w \cdot x_i)w \\ &\quad -(w \cdot x_i)w \cdot x_i + (w \cdot x_i)w \cdot (w \cdot x_i)w \\ &= ||x_i||^2 - 2(w \cdot x_i)^2 + (w \cdot x_i)w \cdot w \\ &= x_i \cdot x_i - (w \cdot x_i)^2 \end{aligned}$$

since $w \cdot w = ||w||^2 = 1$. Add those residuals up across all the vectors:

$$\begin{aligned} MSE(x) &= \tfrac{1}{n}\textstyle\sum_{i=1}^{n} ||x_i||^2 - (w \cdot x_i)^2 \\ &= \tfrac{1}{n}\left(\textstyle\sum_{i=1}^{n} ||x_i||^2 - \sum_{i=1}^{n}(w \cdot x_i)^2\right) \end{aligned}$$

The first summation does not depend on w, so it does not matter for trying to minimize the MSE. To make the MSE small, we need to make the second sum big, i.e., we want to maximize $\frac{1}{n}\sum_{i=1}^{n}(w \cdot x_i)^2$, which we can see is the sample mean of $(w \cdot x_i)^2$. The mean of a square is always equal to the square of the mean plus the variance:

$$\frac{1}{n}\sum_{i=1}^{n}(w \cdot x_i)^2\left(\frac{1}{n}\sum_{i=1}^{n} x_i \cdot w\right)^2 + \mathrm{Var}[w \cdot x_i]$$

3.1.3 Maximizing Variance

Let us maximize the variance. Let us do the algebra in matrix form.

$$\begin{aligned}
\sigma_w^2 &= \tfrac{1}{n}\textstyle\sum_i (x_i \cdot w)^2 \\
&= \tfrac{1}{n}(\mathbf{xw})^T(\mathrm{xw}) \\
&= \tfrac{1}{n}\mathbf{w}^T\mathbf{x}^T\mathbf{xw} \\
&= \mathbf{w}^T\tfrac{\mathbf{x}^T\mathbf{x}}{n}\mathbf{w} \\
&= \mathbf{w}^T\mathbf{vw}
\end{aligned}$$

We want to chose a unit vector w so as to maximize σ_w^2. To do this, we need to make sure that we look at unit vectors, we need to constrain the maximization. The constraint is that $w \cdot w = 1$. The Lagrange multiplier λ, multiplied into the equation, will give us:

$$\begin{aligned}
\mathcal{L}(\mathbf{w}, \lambda) &\equiv \sigma_{\mathbf{w}}^2 - \lambda(\mathbf{w}^T\mathbf{w} - 1) \\
\tfrac{\partial L}{\partial \lambda} &= \mathbf{w}^T\mathbf{w} - 1 \\
\tfrac{\partial L}{\partial \mathbf{w}} &= 2\mathbf{vw} - 2\lambda\mathbf{w}
\end{aligned}$$

Setting the derivatives to zero for the optimal location, we get

$$\mathbf{w}^T\mathbf{w} = 1$$

$$\mathbf{vw} = \lambda\mathbf{w}$$

Thus, desired vector $\mathbf{w}$ is an eigenvector of the covariance matrix $\mathbf{v}$, and the maximizing vector will be the one associated with the largest eigenvalue λ.

Observe $\mathbf{v}$ is a $p \times p$ matrix, thus it will have p different eigenvectors. we know that $\mathbf{v}$ is a covariance matrix, so ti is symmetric and linear algebra tell solve for eigenvectors that must be orthogonal to each other. These eigenvectors of $\mathbf{v}$ are the principal components of the data.

3.2 Clustering Methods

Clustering refers to a very broad set of techniques for finding subgroups, or clusters, in a data set. When we cluster the observations of a data set, we seek to partition them into distinct groups so that the observations within each group are quite similar to each other, while observations in different groups are quite different from each other.

For instance, suppose that we have a set of n observations, each with p features. The n observations could correspond to tissue samples for patients with breast cancer, and the p features could correspond to measurements collected for each tissue sample; these could be clinical measurements, such as tumor stage or grade, or they could be gene expression measurements. We may have a reason to believe that there is some heterogeneity among the n tissue samples; for instance, perhaps there are a few different unknown subtypes of breast cancer. Clustering could be used to find these subgroups. This is an unsupervised problem because we are trying to discover structure — in this case, distinct clusters — on the basis of a data set.

Both clustering and PCA seek to simplify the data via a small number of summaries, but there are some differences:

• PCA looks to find a low-dimensional representation of the observations that explain a good fraction of the variance;

• Clustering looks to find homogeneous subgroups among the observations.

3.2.1 K-Means Clustering

K-means clustering is a simple and elegant approach for partitioning a data set into K distinct, non-overlapping clusters. To perform K-means clustering, we must first specify the desired number of clusters K; then the K-means algorithm will assign each observation to exactly one of the K clusters.

The K-means clustering procedure results from a simple and intuitive mathematical problem. Let $C_1, ..., C_K$ denote sets containing the indices of the observations in each cluster. These sets satisfy two properties:

1. $C_1 \cup C_2 \cup ... \cup C_K = \{1, ..., n\}$. In other words, each observation belongs to at least one of the K clusters.
2. $C_k \cap C_{k'} = \emptyset$ for all $k \neq k'$. In other words, the clusters are non-overlapping: no observation belongs to more than one cluster.

If the ith observation is in the kth cluster, then $i \in C_k$. The idea behind K-means clustering is that a good clustering is one for which the within-cluster variation is as small as possible. The within-cluster variation for cluster C_k is is a measure $W(C_k)$ of the amount by which the observations within a cluster differ from each other. Hence we want to solve the problem

$$\min_{C_1,...,C_K} \left\{ \sum_{k=1}^{K} W(C_k) \right\}.$$

This formula tells that we want to partition the observations into K clusters such that the total within-cluster variation, summed over all K clusters, is as small as possible.

Solving the equation above, we need to define the within-cluster variation. There are many possible ways to define this concept, but by far the most common choice involves squared Euclidean distance. That is, we define

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i/j})^2,$$

where $|C_k|$ denotes the number of observations in the kth cluster. The within-cluster variation for the kth cluster is the sum of all of the pairwise squared Euclidean distances between the observations in the kth cluster, divided by the total number of observations in the kth cluster. Combining the two equations above gives the optimization problem that defines K-means clustering,

$$\min_{C_1,...C_K} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i/ \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i/j})^2 \right\}$$

Now we introduce an algorithm to solve the aove equation, that is, a method to partition the observations into K clusters such that the objective is minimized.

Algorithm. K-Means Clustering

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the cluster assignments stop changing:

   (a) For each of the K clusters, compute the cluster centroid. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.

   (b) Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance).

The above algorithm gauranteed to decrease the value of the objective function at each step. The following identity illustrates the reasoning:

$$\frac{1}{|C_k|}\sum_{i,i\ \in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i.j})^2 = 2\sum_{i\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})^2,$$

where $\bar{x}_{kj} = \frac{1}{|C_k|}\sum_{i\in C_k} x_{ij}$ is the mean for feature j in cluster C_k. In Step 2(a), the cluster means for each feature are the constants that minimize the sum-of-squared deviations, and in Step 2(b), reallocating the observations can only improve the objective function. This means that as the algorithm iterates, the clustering obtained will continually improve until the result no longer changes; the objective function will never increase. In this case, we have reached a local optimum.

3.2.2 Hierarchical Clustering

One potential disadvantage of K-means clustering is that it requires us to pre-specify the number of clusters K. HIerarchical clustering is an alternative approach which does not require that we commit to a particular choice of K. Hierarchical clustering has an added advantage over K-means clustering in that it resutls in an attractive tree-based representation of the observations, called a dendrogram.

We describe bottom-up or agglomerative clustering. This is the most common type of hierarchical clustering, and refers to the fact that a dendrogram (generally depicted as an upside-down tree) is built starting from the leaves and combining clusters up to the trunk.

Here we introduce the hierarchical clustering algorithm.

Algorithm. Hierarchical Clustering

1. Begin with n observations and a measure (such as Euclidean distance) of all the $\binom{n}{2} = n(n-1)/2$ pairwise dissimilarities. Treat each observation as its own cluster.

2. For $i = n, n-1, ..., 2$:

   (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.

   (b) Compute the new pairwise inter-cluster dissimilarities among the $i-1$ remaining clusters.

Complte	Maximal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the largest of these dissimilarities.
Single	Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the smallest of these dissimilarities. Single linkage can result in extended, trailing clusters in which single observations are fused one-at-a-time.
Average	Mean intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the average of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length p) and the centroid for cluster B. Centroid linkage can result in undesirable inversions.

4 GENERALIZED LINEAR MODEL

4.1 Exponential Family

The exponential family can be written in the form

$$p(y;\eta) = b(y)\exp(\eta^T T(y) - a(\eta))$$

and η is called the natural parameter (also called the canonical parameter) of the distribution; while $T(y)$ is the sufficient statistic (for the distribution we consider); and $a(\eta)$ is the log partition function. The quantity $e^{-a(\eta)}$ essentially plays the role of a normalization constant, that makes sure the distribution $p(y;\eta)$ sums/integrates over y to 1. A fixed choice of T, a and b defines a family (or set) of distributrions that is parameterized by η; as we vary η, we then get different distributions within this family.

Write the Bernoulli distribution as

$$\begin{aligned} p(y;\phi) &= \phi^y(1-\phi)^{1-y} \\ &= \exp(y\log\phi + (1-y)\log(1-\phi)) \\ &= \exp\Bigg(\bigg(\log\Big(\frac{\phi}{1-\phi}\Big)\bigg)y + \log(1-\phi)\Bigg). \end{aligned}$$

Thus, the natural parameter is given by $\eta = \log(\phi/(1-\phi))$. Inverting this definition for η by solving for ϕ in terms of η, we obtain $\phi = 1/(1+e^{-\eta})$, which is the sigmoid function! To complete the formulation of the Bernoulli distribution as an exponential family distribution, we also have

$$T(y) = y$$

$$a(\eta) = -\log(1-\phi) = \log(1+e^\eta)$$

$$b(y) = 1$$

This shows that the Bernoulli distribution can be written as above with appropriate chocie of T, a and b.

Let us consider Gaussian distribution.

$$\begin{aligned} p(y;\mu) &= \frac{1}{\sqrt{2\pi}}\exp\Bigg(-\frac{1}{2}(y-\mu)^2\Bigg) \\ &= \frac{1}{\sqrt{2\pi}}\exp\Bigg(-\frac{1}{2}y^2\Bigg)\cdot\exp\Bigg(\mu y - \frac{1}{2}\mu^2\Bigg) \end{aligned}$$

Thus, we see that Gaussian is in the exponential family, with

$$\eta = \mu$$

$$T(y) = y$$

$$a(\eta) = \mu^2/2 = \eta^2/2$$

$$b(y) = (1/\sqrt{2\pi})\exp(-y^2/2).$$

4.2 Constructing GLMs

Suupose you would like to build a model to estimate the number y of customers arriving in your store (or number of page-views on your website) in any given hour, based on certain features x such as store promotions, recent advertising, weather, day-opf-week, etc. We know that the Poisson distribution usually gives a good model for numbers of visitors. Knowing this, how can we come up with a model for our problem? Fortunately, the POisson is an exponential family distribution, so we can apply a GLM.

In general, consider a classification or regression problem where we would like to predict the value of some random variable y as a function of x. To derive a GLM for this problem, we will make the following three assumptions about the conditional distribution of y given x and about our model

1. $y|x;\theta \sim \text{Exp}(\eta)$, i.e., given x and θ, the distribution of y follows some exponential family distribution, with parameter η.
2. Given x, our goal is to predict the expected value of $T(y)$ given x. In most examples, we will have $T(y) = y$, so this means we would like the prediction $h(x)$ output by our learned hypothesis h to satisfy $h(x) = E[y|x]$. (Note that this assumption is satisfied in the choices for $h_\theta(x)$ for both logistic regression and linear regression. For example, in logistic regression, we had $h_\theta(x) = p(y = 1|x;\theta) = 0 \cdot p(y = 0|x;\theta) + 1 \cdot p(y = 1|x;\theta) = E[y|x;\theta]$.)
3. The natuaral parameter η and the inputs x are related linearly: $\eta = \theta^T x$. Or, if η is vector-valued, then $\eta_i = \theta_i^T x$.

4.2.1 Ordinary Least Squarers

Ordinary least squares is a special case of the GLM family of models. Consider the setting where the target variable y (also called the response variable in GLM terminology) is continuous. We model conditional distribution of y given x as a Gaussian $\mathcal{N}(\mu, \sigma^2)$. Thus we let the $\text{Exp}(\eta)$ distribution above be the Gaussian distribution. In the formulation of the Gaussian as an exponential family distribution, we had $\mu = \eta$. Thus, we have

$$\begin{aligned} h_\theta(x) &= E[y|x;\theta] \\ &= \mu \\ &= \eta \\ &= \theta^T x. \end{aligned}$$

The first equality follows from Assumption above; and the second equality follows from the fact that $y|x;\theta \sim \mathcal{N}(\mu, \sigma^2)$, and so its expected value is given by μ; the third equality follows from Assumption 1 (and our earlier derivation showing that $\mu = \eta$ in the formulation of the Gaussian as an exponential family distribution); and the last equality follows from Assumption 3.

4.2.2 Logistic Regression

Now we consider logistic regression. For this case, we are interested in binary classification in the form $y \in \{0, 1\}$. Given that y is binary-valued, it therefore seems natural to choose the Bernoulli family of distributions to model the conditional distribution of y given x. In our formulation of the Bernoulli distribution as an exponential family distribution, we had $\phi = 1/(1 + e^\eta)$. Hence, following a similar derivation as the one for ordinary least squares, we obtain

$$\begin{aligned} h_\theta(x) &= E[y|x;\theta] \\ &= 1/(1 + e^\eta) \\ &= 1/(1 + e^{-\theta^T x}) \end{aligned}$$

This gives us hypothesis functions of the form $h_\theta(x) = 1/(1+e^{-\theta^T x})$. Once we assume that y conditioned on x is Bernoulli, it arises as a consequence of the definition of GLMs and exponential family distributions. The function g given the distribution's mean as a function of the natural parameter $g(\eta) = E[T(y); \eta]$ is called the canonical response function. Its inverse, g^{-1}, is called the canonical link function. Thus, the canonical response function for the Gaussian family is just the identify function; and the canonical response function for the Bernoulli is the logistic function.

4.2.3 Softmax Regression

Consider a classification problem in which the response variable y can take on any one of k values, so $y \in \{1, 2, ..., k\}$. For example, rather than classifying email into the two classes spam or not-spam — which would have been a binary classification problem — we might want to classify it into three classes, such as spam, personal mail, and work-related mail. The response variable is still discrete, but can now take on more than two values. We will thus model it as distributed according to a multinomial distribution.

Let us derive a GLM for modelling this type of multinomial data. Let us begin by writing the multinomial as an exponential family distribution.

To parameterize a multinomial over k possible outcomes, one could use k parameters $\phi_1, ..., \phi_k$ specifying the probability of each of the outcomes. However, these parameters would be redundant, or more formally, they would not be independent (since knowing any $k-1$ of the ϕ_i's uniquely determines the last one, as they must satisfy $\sum_{i-1}^{k} \phi_i = 1$). Thus, we will instead parameterize the multinomial with only $k-1$ parameters, $\phi_1, ..., \phi_{k-1}$, where $\phi_i = p(y = i; \phi)$, and $p(y = k; \phi) = 1 - \sum_{i=1}^{k-1} \phi_i$, but we should keep in mind that this is not a parameter, and that it is fully specified by $\phi_1, ..., \phi_{k-1}$.

To express the multinomial as an exponential family distribution, we will define $T(y) \in \mathbb{R}^{k-1}$ as follows:

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, ..., T(k-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, T(k) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Here we do not have $T(y) = y$; also, $T(y)$ is now a $k-1$ dimensional vector, rather than a real number. We will write $(T(y))_i$ to denote the i-th element of the vector $T(y)$.

An indicator function $1\{\cdot\}$ takes on a value of 1 if its argument is true, and 0 otherwise ($1\{\text{True}\} = 1$, $1\{\text{False}\} = 0$). For example, $1\{2 = 3\} = 0$, and $1\{3 = 5 - 2\} = 1$. Thus we can also write the relationship between $T(y)$ and y as $(T(y))_i = 1\{y = i\}$. Moreover, we have that $E[(T(y))_i] = P(y = i) = \phi_i$.

Now let us show that multinomial is a member of the exponential family.

$$
\begin{aligned}
p(y;\phi) &= \phi_1^{1\{y=1\}}\phi_2^{1\{y=2\}}\dots\phi_k^{1\{y=k\}} \\
&= \phi_1^{1\{y=1\}}\phi_2^{1\{y=2\}}\dots\phi_k^{1-\sum_{i=1}^{k-1}1\{y=i\}} \\
&= \phi_1^{(T(y))_1}\phi_2^{(T(y))_2}\dots\phi_k^{1-\sum_{i=1}^{k-1}(T(y))_i} \\
&= \exp((T(y))_1\log(\phi_1)+(T(y))_2\log(\phi_2)+\cdots+(1-\textstyle\sum_{i=1}^{k-1}(T(y))_i)\log(\phi_k)) \\
&= \exp((T(y))_1\log(\phi_1/\phi_k)+(T(y))_2\log(\phi_2/\phi_k)+\cdots+(T(y))_{k-1}\log(\phi_{k-1}/\phi_k)+\log(\phi_k)) \\
&= b(y)\exp(\eta^TT(y)-a(\eta))
\end{aligned}
$$

where

$$
\begin{aligned}
\eta &= \begin{bmatrix}\log(\phi_1/\phi_k)\\ \log(\phi_2/\phi_k)\\ \vdots \\ \log(\phi_{k-1}/\phi_k)\end{bmatrix} \\
a(\eta) &= -\log(\phi_k) \\
b(y) &= 1.
\end{aligned}
$$

This finishes the formulation of the multinomial as an exponential family distribution. The link function is given, for $i=1,...,k$, there is

$$\eta_i = \log\frac{\phi_i}{\phi_k}.$$

For convenience, we have also defined $\eta_k = \log(\phi_k/\phi_k) = 0$. To invert the link function and derive the response function, we therefore have that

$$e^{\eta_i} = \frac{\phi_i}{\phi_k}$$

$$\phi_k e^{\eta_i} = \phi_i$$

$$\phi_k\sum_{i=1}^k e^{\eta_i} = \sum_{i=1}^k \phi_i = 1$$

This implies that $\phi_k = 1/\sum_{i=1}^k e^{\eta_i}$, which can be substituted back into above equations and we have response function

$$\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}}$$

This function mapping from the η's to the ϕ's is called the softmax function.

To complete our model, we use Aasumption 3 that the η_i's are linearly related to the x's. Thus, we have $\eta_i = \theta_i^Tx$ (for $i=1,...,k-1$), where $\theta_1,...,\theta_{k-1}$ are the parameters of our model. We can define $\theta_k=0$, so that $\eta_k = \theta_k^Tx = 0$. Hence, our model assumes that the conditional distribution of y given x is given by

$$
\begin{aligned}
p(y=i|x;\theta) &= \phi_i \\
&= \frac{e^{\eta_i}}{\sum j=1^k e^{\eta_j}} \\
&= \frac{e^{\theta_i^Tx}}{\sum_{j=1}^k e^{\theta_j^Tx}}
\end{aligned}
$$

This model, which applies to classification problems where $y \in \{1, ..., k\}$, is called softmax regression. It is a generalization of logistic regression. Our hypothesis will output

$$\begin{aligned}
h_\theta(x) &= E[T(y)|x;\theta] \\
&= E\left[\begin{pmatrix} 1\{y=1\} \\ 1\{y=2\} \\ \vdots \\ 1\{y=k-1\} \end{pmatrix} |x;\theta\right] \\
&= \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \frac{\exp(\theta_2^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_1 k-1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \end{bmatrix}
\end{aligned}$$

In other words, our hypothesis will output the estimated probability that $p(y = i|x; \theta)$, for every value of $i = 1, ..., k$. Even though $h_\theta(x)$ as defined above is only $k - 1$ dimensional, clearly $p(y = k|x; \theta)$ can be obtained as $1 - \sum_{i=1}^{k-1} \phi_i$.

Last, let us discuss parameter fitting. Similar to our original derivation of ordinary least squares and logistic regression, if we have a training set of m examples $\{(x^{(i)}, y^{(i)}); i = 1, ..., m\}$ and would like to learn the parameters θ_i of this model, we would begin by writing down the log-likelihood

$$\begin{aligned}
l(\theta) &= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)};\theta) \\
&= \sum_{i=1}^m \log \prod_{l=1}^k \left(\frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \right)
\end{aligned}$$

To obtain the second equality, we used the definition for $p(y|x; \theta)$ given in updated conditional distribution of y. We can now obtain the maximum likelihood estimate of the parameters by maximizing $l(\theta)$ in terms of θ, using a method such as gradient ascent or Newton's method.

5 RESAMPLING AND MODEL SELECTION

The objective for model selection is that we often times have multiple models ahead of us and for each of them there is a lot of tuning needed to perform high testing set accuracy. Cross validation is a method which tries to select the best model from a given set of models. In model selection, we assume that quality measure is predictive performance. "Set of models" can simply mean "set of different parameter values."

For example, we can consider a model selection problem for SVM. The SVM is a family of models indexed by the margin parameter γ and the kernel parameters σ. Our goal is to find a value of (γ, σ) for which we can expect small generalization error. We can include (γ, σ) into the optimization problem, i.e. train by minimizing over α and (γ, σ). This leads to a phenomenon called overfitting: the classifier adapts too closely to specific properties of the training data, rather than the underlying distsribution. For illustration, plotted graphs will have training error decrease as model gets more and more complicated yet testing error may decrease first but increase later. If classifier can adapt too well to the data, there may be small training error, but possibly large testing error. If classifier can hardly adapt at all, there is large training error and also testing error. An ideal model would lie somewhere in between.

5.1 Cross Validation

First, we randomly split data into three sets: training, validation and test data. Second, label training classifier on training data for different values of parameters, say γ. Third, evaluate each trained classifier on validation data, i.e., compute error rate on validation data. Fourth, select the value of parameters with the lowest error rate from validation data. Last, use the parameter from previous step to compute error rate for the test data.

The quality measure by which we are comparing different classifiers $f \cdot; \gamma)$ for different apramemeter values γ is the risk

$$R(f(\cdot;\gamma)) = \mathbb{E}[L(y, f(x;\gamma))].$$

Since we do not know the true risk, we estimate it from data as $\hat{R}(f \cdot; \gamma))$. We always have to assume that the classifier is better adapted to any data used to select it than to actual data distribution. The final model, ideally, would adapt classifier to both training and validation data. If we estimate error rate on this data, we will in general underestimate it. The procedure for Cross Vlidation is as follows:

1. For each value in parameter $\gamma_1, ..., \gamma_m$, train a classifier $f(\cdot, \gamma_j)$ on the training set.
2. Use the validation set to estimate $R(f(\cdot;\gamma_j))$ as the empirical risk

$$\hat{R}(f(x;\gamma_j)) = \frac{1}{n_v} \sum_{i=1}^{n_v} L(\tilde{y}_i, f(\tilde{x}_i, \gamma_j)),$$

   while n_v is the size of the validation set.
3. Select the value γ^* which achieves the smallest estimated error.
4. Re-train the classifier with parameter γ^* on all data except the test set (i.e. on training + validation data).
5. Report error estimate $\hat{R}(f(\cdot;\gamma^*))$ computed on the test set.

5.2 K-Fold Cross Validation

The idea is that each of the error estimates computed on validation set is computed from a single example of a trained classifier. We want to improve this estimates? The strategy is to set aside the test set. We want to split the remaining data into K blocks Use each block in turn as validation set. Perform cross validation and average the results over all K combinations. This method is called $K-fold cross validation.

To estimate the risk of a classifier $f(\cdot, \gamma_j)$, we operate the following procedure:

1. Split data into K equally sized blocks.
2. Train an instance $f_k(\cdot, \gamma_j)$ of the classifier, using all blocks except block k as training data.
3. Compute the cross validation estimate

$$\hat{R}_{CV}(f(\cdot, \gamma_j)) := \frac{1}{K} \frac{1}{|\text{block } k|} \sum_{(\tilde{x}, \tilde{y})}$$

6 NON-LINEAR REGRESSION

6.1 Polynomial

To be replace traditional linear model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

we consider a polynomial function

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \cdots + \beta_d x_i^d + \epsilon_i,$$

where ϵ_i is the error term. This approach is known as polynomial regression.

6.2 Step Function

Using polynomial functions of the features as predictors in a linear model imposes a global structure on the non-linear of X. Instead we can also use step functions in order to avoid imposing such a global structure. We break the range of X into bins, and fit a different constant in each bin. This amounts to converting a continuous variable into an ordered categorical variable.

In details, we create cutpoints $c_1, c_2, ..., c_K$ in the range of X, and then construct $K+1$ new variables

$$\begin{array}{rcl} C_0(X) & = & I(X < c_1), \\ C_1(X) & = & I(c_1 \leq X < c_2), \\ C_2(X) & = & I(c_2 \leq X < c_3), \\ & \vdots & \\ C_{K-1}(X) & = & I(c_{K-1} \leq X < c_K), \\ C_K(X) & = & I(c_K \leq X), \end{array}$$

where $I(\cdot)$ is an indicator function that returns a 1 if the condition is true, and returns a 0 otherwise. These dummy variables are created to sum to 1, that is, for any X, $C_0(X) + C_1(X) + \cdots + C_K(X) = 1$.We can then use least squares to fit a linear model using $C_1(X), C_2(X), ..., C_K(X)$ as predictors:

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \cdots + \beta_K C_K(x_i) + \epsilon_i.$$

6.3 Basis Functions

Polynomial and piecwise-constant regression models are special cases of a basis function approach. The idea is to have at hand a family of functions or transformations that can be applied to a variable $X : b_1(X), b_2(X), ..., b_K(X)$. Instead, we fit the model

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + \cdots + \beta_K b_K(x_i) + \epsilon_i.$$

Note that the basis functions $b_1(\cdot)$, $b_2(\cdot)$, $\ldots$, $b_K(\cdot)$ are fixed and known. For polynomial regression, the basis functions are $b_j(x_i) = x_i^j$, and for piece wise constant functions they are $b_j(x_i) = I(c_j \leq x_i < c_{j+1})$.

6.4 Regression Splines

6.4.1 Piecewise Polynomials

Instead high-degree polynomial over the entire range of X, piecewise polynomial regression involves fitting separate low-degree polynomials over different regions of X. For example, a piecewise cubic polynomial works by fitting a cubic regression model of the form

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i,$$

where the coefficients β_0, β_1, β_3 differ in different parts of the range of X. The points where the coefficients chane are called knots.

6.4.2 Constraints and Splines

Sometimes the scatter plot versus the non-linear plot for models look very much non-comparable. To fix this problem,m we can adjust our model by fitting a piecewise polynomial under the constraint that the fitted curve must be continuous. Such way the non-linear plot will look continuous and more natural. In doing such, we are reducing degrees of freedom for partial piecewise polynomials.

7 TREE CLASSIFIERS

This chapter we describe tree-based methods for regression and classification. These involve stratifying or segmenting the predictor space into a number of simple regions. We introduce bagging, random forests, and boosting. Each of these approaches involves producing multiple trees which are then combined to yield a single consensus prediction.

7.1 Regression Tree

How to build a regression tree? There are two steps.

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, ..., X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j

For instance, suppose that in Step 1 we obtain two regions, R_1 and R_2, and that the response mean of the training observations in the first region is 10, while the response mean of the training observations in the second region is 20. Then for a given observation $X = x$, if $x \in R_1$ we will predict a value of 10, and if $x \in R_2$ we will predict a value of 20.

The goal is to find boxes $R_1, ..., R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box. We apply a top-down approach that is known as recursive binary splitting. This method begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.

To perform such recursive binary splitting, we first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $X|X_j \geq s\}$ leads to the greatest possible reduction in RSS. In details, for any j and s, we define the pair of half-planes

$$R_1(j,s) = \{X|X_j < s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\},$$

and we seek the value of j and s that minimize the equation

$$\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

where $\hat{y}_{R_1}$ is the mean response for the training observations in $R_1(j,s)$, and $\hat{y}_{R_2}$ is the mean response for the training observations in $R_2(j,s)$.

Next, we repeat this process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions. However, instead of splitting the entire predictor spacewe split one of the two previously identified regions.

7.2 Pruning

The process described above many produce good predictions on training set, but is likely to overfit the data, leading to poor testing set performance. This is because the resulting tree might be too complex. A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias.

A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree. How do we determine the best way to prune the tree? We want to select a subtree that leads to the lowest test error. We want to select efficiently a small set of subtrees for consideration.

Cost complexity pruning — also known as weakest link pruning — gives us a way to do just this.

Algorithm Building a Regression Tree

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, ..., K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α. Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

is as small as possible. Here $|T|$ indicates the number of terminal noces of the tree T, R_m is the rectangle corresponding to the mth terminal noce, and $\hat{y}_{R_m}$ is the predicted response associated with R_m — that is, the mean of the training observations in R_m. The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.

7.2.1 Classification Trees

A classification tree is very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one. For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. We are interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region.

The task of growing a classification tree is similar to the task of growing a regression tree. Since we plan to assign an observation in a given region to the most commonly occurring class of training observations in that region, the classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}).$$

In this case, $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. Howeut that classification error is not sufficiently sensitive for tree-growing.

The Gini index is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

a measure of total variance across the K classes. It is not hard to see that the Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. For this reason the Gini index is referred to as a measure of node purity — a small value indicates that a node contains predominantly observations from a single class.

An alternative to the Gini index is entropy, given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

Since $0 \leq \hat{p}_{mk} \leq 1$, it follows that $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$. One can show that the entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all near zero or near one. Therefore, like the Gini index, the entopy will take on a small value if the mth node is pure. In fact, it turns out that the Gini index and the entropy are quite similar numerically.

7.2.2 Advantages and Disadvantages of Trees

Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression! Some people that decision trees more closely mirror human decision-making than do regression and classification hes seen in previous sectiTrees can be displayed graphically, and are easily interpreted evespecially if they are small). Trees can easily handle qualitative predictors without the need to create variables.

Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen before. Additionally, trees can be very non-robust. In otherords, a small change in the data can cause a large change in the final estimated tree.

7.3 Bagging

Bootstrap is an extremely powerful idea. Here we see that the bootstrap can be used in a completely different context, such as decision trees.

Decision trees suffer from high variance. This means that if we split the training data into two parts at random and fit a decision tree the results that we get could be quite different. Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

Given a set of n independent observations $X_1, ..., X_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n. In other words, averaging a set of observations reduces variance. Hence, a natural way to reduce the variance and hence increase the prediction accuracy of statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predicdtions. We calculate $\hat{f}^1(x), \hat{f}^2(x), ..., \hat{f}^B(x)$ using B separate training sets, and average them in order to obtain a single low-variance statistical learning model, given by

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x).$$

In practice, this is not accessable because we need multiple training sets. Instead, we can bootstrap, by taking repeated samples from the single training data set. In this approach we generate B different bootstrapped training data sets. Then we train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, and finally average all the predictions, to obtain

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

This is called bagging.

7.3.1 Out-of-bag (OOB)

In general each bagged tree makes of two-thirds of the observations. The remaining one-third of the observations not used to fit given bagged tree are referred to as the out-of-bag (OOB) observations. We can predict the response for the ith observation using each of the trees in which that observation was OOB. This way, each prediction results an overall OOB MSE for a regression problem or classification error for a classification problem.

7.4 Random Forests

Random forests provide improvement over bagged trees by way of a small tweak that decorrelates the trees. As in bagging decision trees on bootstrapped training sample. In the process of building decision trees, a split in a tree occurs each time; and a random sample of m predictors is chosen as split candidates from the full set of p predictors. In other words, in building a random forest, at each split in the tree, the algorithm is not allowed to consider a majority of the vailable predictors.

Random forests force each split to consider only a subset of the predictors. Therefore, on average $(p-m)/p$ of the splits will not consider the strong predictor, and so other predictors will have more of a chance. This process, random forests, can also be thought of as decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable. Thus, the main difference between bagging and random forests is the choice of predictor subset size m.

7.5 Boosting

Boosting is another approach for improving the predictions resulting from a decision tree. Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. Boosting works in a similar way as bagging, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.

Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly. Given the current model, we fit a decision tree to the residuals from the model. That is, we fit a tree using the current residuals, rather than the outcome Y, as the response. We then add this new decision tree into the fitted function in order to update the residuals. Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.

By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals. In general, statistical learning approaches that learn slowly tend to perform well. Note in boosting, unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown.

Boosting classification trees proceeds in a similar but slightly more complex way. Boosting has three tuning parameters:

1. The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns.
3. The number d of splits in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a stump, consisting of a single split.

Algorithm. Boosting for Regression.

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, ..., B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   (c) Update the residuals,

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

8 SUPPORT VECTOR MACHINE

8.1 Hyperplanes

We introduce hyperplanes as a foundation to the theoretical framework of SVM. A hyperplane in $\mathbb{R}^d$ is a linear subspace of dimension $(d-1)$. For $d=2$, the hyperplane is a line; and for $d=3$, it is a plane. A hyperplane H can be represented by a normal vector. The hyperplane with normal vector v_H is the set

$$H = \{x \in \mathbb{R}^d | < x, v_H >= 0\}.$$

We can also determine which side of plane are we on. The projection of x onto the direction of v_H has length $< x, v_H >$ measured in units of v_H, i.e. length $< x, v_H > /||v_H||$ in the units of the coordinates. Based on cosine rule $\cos\theta = \frac{<x,v_H>}{||x||\cdot||v_H||}$, the distance of x from the plane is given by

$$d(x,H) = \frac{< x, v_H >}{||v_H||} = \cos\theta \cdot ||x||.$$

We can then decide the side of the plane x is on using

$$\mathrm{sgn}(\cos\theta) = \mathrm{sgn} < x, v_H >$$

An affine hyperplane H_W is a hyperplane translated (shifted) by a vector w, i.e. $H_w = H + w$. We choose w in the direction of v_H, i.e. $w = c \cdot V_H$ for $c > 0$. Then we decide which side of plane we are on by computing

$$\mathrm{sgn}(< x - w, v_H >) = \mathrm{sgn}(< x, v_H > -c < v_H, v_H >) = \mathrm{sgn}(< x, v_H > -c||v_H||^2)$$

If v_H is a unit vector, we can use $\mathrm{sgn}(< x - v_H >, v_H) = \mathrm{sgn}(< x, v_H > -c)$.

8.2 Linear Classifier

A linear classifier is a function of the form

$$f_H(x) := \mathrm{sgn}(< x, v_H > -c),$$

where $v_H \in \mathbb{R}^d$ is a vector and $c \in \mathbb{R}_+$. Note that we usually assume v_H to be a unit vector. If it is not, f_H still defines a linear classifier, but c describes a shift of a different length. We have the following definition. Two sets $A, B \in \mathbb{R}^d$ are called linearly separable if there is an affine hyperplane H which separates them, i.e. which satisfies

$$< x, v_H > -c = \begin{cases} < 0 & \text{if } x \in A \\ > 0 & \text{if } x \in B \end{cases}$$

8.3 Maximum Margin

Suppose we have a classification problem with response $Y = -1$ or $Y = 1$. If the class can be separated, most likely, there will be an infinite number of hyperplanes separating the classes. The idea is to draw the largest possible empty margin around the hyperplane. Out of all possible hyperplanes that separate the two classes, choose the one such that distance to closest point in each class is maximal. This distance is called the margin. The classifier should cut off as little probability mass as possible from either distribution. Such method is called optimal generalization. For occasions that we cannot or do not know the density contour, we would use convex hull as a substituion. If C is a set of points containing all points in C is called the convex hull of C, denoted $conv(C)$. The coner points of the convex set are called extreme points. Every point x in a convex set can be represented as a convex combination of the extreme points $\{e_1, ..., e_M\}$. There are weights $\alpha_1, ..., \alpha_m \in \mathbb{R}_+$ such that

$$x = \sum_{i=1}^{m} \alpha_i e_i \text{ and } \sum_{i=1}^{m} \alpha_i = 1$$

The coefficients α_i in the above equation are called barycentric coordinates of x.

The key idea is the following. A hyperplane separates two classes if and only if it separates their convex hull. Before we proceed, let us introduce some definitions. The distance between a point x and a set A the Euclidean distance between x and the closest point in A:

$$d(x, A) := \min_{y \in A} ||x - y||$$

In particular, if $A = H$ is a hyperplane, $d(x, H) := \underset{y \in H}{min} ||x - y||$. The margin of a classifier hyperplane H given two training classes X_- and X_+ is the shortest distance between the plane and any point in either set:

$$\text{margin} = \min_{x \in X_- \cup X_+} d(x, H)$$

Equivalently, we write the shortest distance to either of the convex hulls is given by

$$\text{margin} = \min\{d(H, \text{conv}(X_-), d(H, \text{conv}(X_+)\}$$

For normal vector v_H, we have the following to identify different signs

$$< v_H, x > -c \begin{cases} > 0 & x \text{ on positive side} \\ < 0 & x \text{ on negative side} \end{cases}$$

The scalar $c \in \mathbb{R}$ specifies shift (plane through origin if $c = 0$). Then the demand is $< v_H, x > -c > 1$ or < -1 with $\{-1, 1\}$ on the right works for any margin. The size of margin is determined by $||v_h||$. To increase margin, we scale down v_H. The concept of margin applies only to training, not to classification. Classification works as for any linear classifier. For a test point x:

$$y = \text{sign}(< v_H, x > -c)$$

For n training points $(\tilde{x}_i, \tilde{y}_i)$ with labels $\tilde{y}_i \in \{-1, 1\}$, solve optimization problem

$$\min_{v_H, c} ||v_H|| \text{ such that } \tilde{y}_i(< v_H, \tilde{x}_i > -c) \geq 1 \text{ for } i = 1, ..., n$$

The classifier obtained by solving this optimization problem is called a support vector machine. We can project a vector x (say, an observation from training data) onto the direction of v_H and obtain vector x_V. If H has no offset ($c = 0$), the Euclidean distance of x from H is

$$d(x, H) = ||x_v|| = \cos\theta \cdot ||x||.$$

It does not depend on the length of v_H. The scalar product $< x, v_H >$ does increase if the length of v_H increases. To compute the distance $||X_V||$ from $< x, v_H >$, we have to scale out $||v_H||$:

$$||x_V|| = \cos\theta \cdot ||x|| = \frac{< x, v_H >}{||v_H||}$$

8.4 Kernels

For kernels, we have the following motivation. First, we assume there is a linear decision boundary. Next, there exist perceptrons, which are linear separability and placement of boundary rather arbitrary. For example, the SVM uses the scalar product $< x, \tilde{x}_i >$ as a measure of similarity between x and $\tilde{x}_i$, and of distance to the hyperplane. Since the scalar product is linear, the SVM is a linear method. By using a nonlinear function instead, we can make the classifier nonlinear.

More precisely, scalar product can be regarded as a two-argument function

$$< \cdot, \cdot >: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

We will replace this function with a function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and substitute

$$k(x, x') \text{ for every occurrence of } < x, x' >$$

in the SVM formulae. Under certain conditions on k, all optimization/classification results for the SVM still hold. Functions that satisfy these conditions are called kernel functions.

8.4.1 RBF

RBF Kernel, which takes the following form,

$$k_{RBF}(x, x') := \exp\big(-\frac{||x - x'||_2^2}{2\sigma^2}\big)$$

is called an RBF kernel (i.e. radial basis function). The paramter σ is called bandwith. Other names for k_{RBF} are Gaussian kernel, squared-exponential kernel. If we fix x', the function $k_{RBF}(\cdot, x')$ is up to scaling a spherical Gaussian density on $\mathbb{R}^d$, with mean x' and standard deviation σ.

To define a kernel, we have to define a fucntion of two arguments and prove that it is a kernel. This is done by checking a set of necessary and sufficient conditions known as "Mercer's Theorem". In practice, the data analyst does not define a kernel, but tries some well-known standard kernels until one seems to work. MOst common choises are RBF kernel, or the linear kernel, $k_{SP}(x, x') =< x, x' >$, i.e., the standard. One a kernel is chosen, the classifier can be trained by solving the optimization problem using standard software. SVM software packages include implementations of most common kernels.

8.4.2 Definition: Kernel Function

A function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called a kernel on $\mathbb{R}^d$ if there is some function $\phi : \mathbb{R}^d \to \mathcal{F}$ into some space $\mathcal{F}$ with scalar product $< \cdot, \cdot >_{\mathcal{F}}$ such that

$$k(x, x') =< \phi(x), \phi(x'))_{\mathcal{F}} \text{ for all } x, x' \in \mathbb{R}^d.$$

In other words, k is a kernel if it can be interpreted as a scalar product on some other space. If we substitute $k(x, x')$ for $< x, x' >$ in all SVM equations, we implicitly train a linear SVM on the space $\mathcal{F}$. The SVM still wroks and it just uses scalar products on another space.

The mapping ϕ has to transform the data into data on which a linear SVM works well. This is usually achieved by choosing $\mathcal{F}$ as a higher-dimensional space than $\mathbb{R}^d$. In previous example, we have to know what the data looks like to choose ϕ. The solution is to choose high dimension h for $\mathcal{F}$, to choose components ϕ_i of $\phi(x) = (\phi_1(x), ..., \phi_h(x))$ as different nonlinear mappings. If two points differ in $\mathbb{R}^d$, some of the nonlinear mappings will amplify differences. The RBF kernel is an extreme case. The function k_{RBF} can be shown to be a kernel, however: $\mathcal{F}$ is infinite-dimensional for this kernel.

8.4.3 Mercer's Theorem

A mathematical result called Mercer's Theorem states that, if the function k is positive, i.e.,

$$\int_{\mathbb{R}^d \times \mathbb{R}^d} k(x, x') f(x) f(x') dx dx' \geq 0$$

for all functions f, then it can be written as

$$k(x, x') = \sum_{j=1}^{\infty} \lambda_j \phi_j(x) \phi_j(x').$$

The ϕ_j are functions $\mathbb{R}^d \to \mathbb{R}$ and $\lambda_i \geq 0$. This means the possibly infinite vector $\phi(x) = (\sqrt{\lambda_1}\phi(x), \sqrt{\lambda_2}\phi_2(x), ...)$ is a feature map.

Many linear machine learning and statistics algorithms can be "kernelized". The only condition are: (1) the algorithm uses a scalar product, and (2) in all relevant equations, the data (and all other elements of $\mathbb{R}^d$) appear only inside a scalar product. This approach to making algorithms non-linear is known as the "kernel trick". It is an optimization problem. Consider

$$\min_{v_H, c} ||v_H||^2_{\mathcal{F}} + \gamma \sum_{i=1}^{n} \xi^2 \text{ such that } y_i(< v_H, \phi(\tilde{x}_i) > -c) \geq 1 - \xi_i \text{ and } \xi \geq 0$$

Note: v_H lives in $\mathcal{F}$, and $|| \cdot ||_{\mathcal{F}}$ and $< \cdot, \cdot >_{\mathcal{F}}$ are norm and scalar product on $\mathcal{R}$. We can transform and solve as a dual optimization problem

$$\max_{\alpha \in \mathbb{R}^n} W(\alpha) := \sum_{j=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j (k(\tilde{x}_i, \tilde{x}_j) + \frac{1}{\gamma} \mathbb{I}\{i = j\})$$

$$\text{such that } \sum_{i=1}^{n} y_i \alpha_i = 0 \text{ and } \alpha_i > 0$$

Then the Classifier is $f(x) = \text{sgn}\big(\sum_{i=1}^{n} \tilde{y}_i \alpha_i^* k(\tilde{x}_i, x) - c\big)$.

8.5 Support Vectors

The extreme points of the convex hulls which are closest to the hyperplane are called the support vectors. There are at least two support vectors, one in each class. The maximum-margin criterion focusses all attention to the area closest to the decision surface. Small changes in the support vectors can result in significant changes of the classifier. In practice, the approach is combined with "slack variables" to permit overlapping classes. As a side effect, slack variables soften the impact of changes in the support vectors.

To solve SVM optimization problem

$$\min_{v_H,c} ||v_H|| \text{ such that } \tilde{y}_i(< v_H, \tilde{x}_i > -c) \geq 1 \text{ for } i = 1, ..., n$$

is difficult, because the constraint is a function. It is possible to transform this problem into a problem which seems more complicated, but has simpler constraints:

$$\max_{\alpha \in \mathbb{R}^n} W(\alpha) := \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \alpha_i \alpha_j \tilde{y}_i \tilde{y}_j < \tilde{x}_i, \tilde{x}_j >$$

$$\text{such that } \sum_{i=1}^{n} \tilde{y}_i \alpha_i = 0 \text{ while } \alpha_i \geq 0 \text{ for } i = 1, ..., n$$

This is called the optimization problem dual to the minimization problem above. It is usually derived using Lagrange multipliers. We will use a more geometric argument.

Many dual relations in convex optimization can be traced back to the following fact: The closest distance between a point x and a convex set A is the maximum over the distances between x and all hyperplanes which separate x and A, mathematically,

$$d(x, A) = \sup_{H \text{ separating}} d(x, H)$$

8.6 Optimization

8.6.1 Optimization Problems

An optimization problem for a given function $f : \mathbb{R}^d \to \mathbb{R}$ is a problem of the form

$$\min_x f(x)$$

which we read as "find $x_0 = \arg\min_x f(x)$". A constrained optimization problem adds additional requirements on x

$$\min_x f(x) \text{ subject to } x \in G,$$

where $G \subset \mathbb{R}^d$ is called the feasible set. The set G is often defined by equation, e.g.,

$$\min_x f(x) \text{ subject to } g(x) \geq 0$$

The equation g is called a constraint. For optimization problems, we discuss global and lcoal minimum. In $\mathbb{R}^d$, $\bigtriangledown f(x) = 0$ and $H_f(x) = \left(\frac{\partial f}{\partial x_i \partial x_j}\right)_{i,j=1,2,...,n}$ are positive definite.

8.6.2 Gradient Descent

Gradient Descent searches for a minimum of f.

1. Start with some point $x \in \mathbb{R}$ and fix a precision $\epsilon > 0$.
2. Repeat for $n = 1, 2, ...$, there is
$$x_{n+i} := x_n - f'(x_n)$$
3. Terminate when $|f'(x_n)| < \epsilon$.

8.6.3 Newton's Method

Newton's method searches for a root of f, i.e., it solves the equation $f(x) = 0$.

1. Start with some point $x \in \mathbb{R}$ and fix a precision $\epsilon > 0$.
2. Repeat for $n = 1, 2, ...$, there is
$$x_{n+i} := x_n - f(x_n)/f'(x_n)$$
3. Terminate when $|f(x_n)| < \epsilon$.

We can also use Newton's Method for minimization by applying it to solve $f'(x) = 0$.

1. Start with some point $x \in \mathbb{R}$ and fix a precision $\epsilon > 0$.
2. Repeat for $n = 1, 2, ...$, there is
$$x_{n+i} := x_n - f'(x_n)/f''(x_n)$$
3. Terminate when $|f'(x_n)| < \epsilon$.

8.6.4 Karush-Kuhn-Tucker

The idea is the following. We want to decompose $\bigtriangledown f$ into a component $(\bigtriangledown f)_s$ in the set $\{x|g(x) = 0\}$ and a remainder $(\bigtriangledown f)_\perp$. The two components are orthogonal. If f_g is minimal within $\{x|g(x) = 0\}$, the component within the eset vanies. The remainder need not vanish. The consequence is that we need to solve for a criterion for $(\bigtriangledown f)_g = 0$. If $(\bigtriangledown f)_g = 0$, then $\bigtriangledown f$ is orthogonal to the set $g9x) = 0$. Since gradients are orthogonal to contours, and the set is a contour of g, $\bigtriangledown_g$ is also orthogonal to the set. Hence, at a minimum of f_g, the two gradients point in the same direction: $\bigtriangledown f + \lambda \bigtriangledown g = 0$ for some scalar $\lambda \neq 0$.

The optimization problem with inequality constraints

$$\min f(x) \text{ subject to } g(x) \leq 0$$

can be solved by solving

$$\left.\begin{array}{rcl} \bigtriangledown f(x) & = & -\lambda \bigtriangledown g(x) \\ \lambda g(x) & = & 0 \\ g(x) & \leq & 0 \\ \lambda & \geq & 0 \end{array}\right\} \text{system of } d+1 \text{ equations for } d+1 \text{ variables } x_1, ..., x_D, \lambda$$

These conditions are known as the Karush-Kuhn-Tucker (KKT) conditions.

9 NEURO-NETWORK

9.1 A Neuron

The area of Neural Networks has originally been primarily inspired by the goal of modeling biological neural systems, but has since diverged and become a matter of engineering and achieving good results in Machine Learning tasks. Nonetheless, we begin our discussion with a very brief and high-level description of the biological system that a large portion of this area has been inspired by.

The basic computational unit of the brain is a neuron. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately $10^{14} - 10^{15}$ synapses.Each neuron receives input signals from its dendrites and produces output signals along its (single) axon. The axon eventually branches out and connects via synapses to dendrites of other neurons. In the computational model of a neuron, the signals that travel along the axons (e.g. x_0) interact multiplicatively (e.g. w_0x_0) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g. w_0). The idea is that the synaptic strengths (the weights w) are learnable and control the strength of influence (and its direction: excitory (positive weight) or inhibitory (negative weight)) of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, we assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. Based on this rate code interpretation, we model the firing rate of the neuron with an activation function f, which represents the frequency of the spikes along the axon. Historically, a common choice of activation function is the sigmoid function σ since it takes a real-valued input (the signal strength after the sum) and squashes it to range between 0 and 1. We will see details of these activation functions later in this section. In other words, each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function), in this case the sigmoid $\sigma(x) = 1/(1 + e^{-x})$.

9.2 Neuron as Linear Classifier

The mathematical form of the model Neuron's forward computation might look familiar to you. As we saw with linear classifiers, a neuron has the capacity to "like" (activation near one) or "dislike" (activation near zero) certain linear regions of its input space. Hence, with an appropriate loss function on the neuron's output, we can turn a single neuron into a linear classifier:

Binary Softmax Classifier. For example, we can itnerpret $\sigma(\sum_i w_i x_i + b)$ to be the probability of one of the classes $P(y_i = 1|x_k; w)$. The probability of the other class would be $P(y_i = 0|x_i; w) = 1 - P(y_i = 1|x_i; w)$, since they must sum to one. With this interpretation, we can formulate the cross-entropy loss as we have seen in the Linear Classification section, and optimizing it would lead to a binary Softmax classifier (also known as logistic regression). Since the sigmoid function is restricted to be between 0-1, the predictions of this classifier are based on whether the output of the neuron is greater than 0.5.

Binary SVM Classifier. Alternatively, we could attach a max-margin hinge loss to the output of the neuron and train it to become a binary Support Vector Machine.

Regularization Interpretation. The regularization loss in both SVM/Softmax cases could in this biological view be interpreted as gradual forgetting, since it would have the effect of driving all synaptic weights ww towards zero after every parameter update.

9.3 Activation Functions

Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions you may encounter in practice:

9.3.1 Sigmoid

The sigmoid non-linearity has the mathematical form $\sigma(x) = 1/(1 + e^{-x})$ and is shown in the image above on the left. As alluded to in the previous section, it takes a real-valued number and "squashes" it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1). In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks:

(1) Sigmoids saturate and kill gradients. A very undesirable property of the sigmoid neuron is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Recall that during backpropagation, this (local) gradient will be multiplied to the gradient of this gate's output for the whole objective. Therefore, if the local gradient is very small, it will effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. Additionally, one must pay extra caution when initializing the weights of sigmoid neurons to prevent saturation. For example, if the initial weights are too large then most neurons would become saturated and the network will barely learn.

(2) Sigmoid outputs are not zero-centered. This is undesirable since neurons in later layers of processing in a Neural Network (more on this soon) would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive (e.g. $x > 0$ elementwise in $f = w^x + b$), then the gradient on the weights w will during backpropagation become either all be positive, or all negative (depending on the gradient of the whole expression f). This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights. However, notice that once these gradients are added up across a batch of data the final update for the weights can have variable signs, somewhat mitigating this issue. Therefore, this is an inconvenience but it has less severe consequences compared to the saturated activation problem above.

9.3.2 Tanh

The tanh non-linearity is shown on the image above on the right. It squashes a real-valued number to the range $[-1, 1]$. Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity. Also note that the tanh neuron is simply a scaled sigmoid neuron, in particular the following holds: $\tanh(x) = 2\sigma(2x) - 1$

9.3.3 ReLU

The Rectified Linear Unit has become very popular in the last few years. It computes the function $f(x) = \max(0, x)$. In other words, the activation is simply thresholded at zero (see image above on the left). There are several pros and cons to using the ReLUs:

(1) (+) It was found to greatly accelerate (e.g. a factor of 6 in Krizhevsky et al. http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.

(2) (+) Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.

(3) (-) Unfortunately, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as

40% of your network can be "dead" (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. With a proper setting of the learning rate this is less frequently an issue.

9.3.4 Leaky ReLU

Leaky ReLUs are one attempt to fix the "dying ReLU" problem. Instead of the function being zero when x < 0, a leaky ReLU will instead have a small negative slope (of 0.01, or so). That is, the function computes $f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$ where α is a small constant. Some people report success with this form of activation function, but the results are not always consistent. The slope in the negative region can also be made into a parameter of each neuron, as seen in PReLU neurons, introduced in Delving Deep into Rectifiers, by Kaiming He et al., 2015 https://arxiv.org/abs/1502.01852. However, the consistency of the benefit across tasks is presently unclear.

9.3.5 Maxout

Other types of units have been proposed that do not have the functional form $f(w^T x + b)$ where a non-linearity is applied on the dot product between the weights and the data. One relatively popular choice is the Maxout neuron (introduced recently by Goodfellow et al. http://www-etud.iro.umontreal.ca/~goodfeli/maxout.html) that generalizes the ReLU and its leaky version. The Maxout neuron computes the function $\max(w_1^T x + b_1, w_2^T x + b_2)$. Notice that both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU we have $w_1, b_1 = 0$). The Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU). However, unlike the ReLU neurons it doubles the number of parameters for every single neuron, leading to a high total number of parameters.

This concludes our discussion of the most common types of neurons and their activation functions. As a last comment, it is very rare to mix and match different types of neurons in the same network, even though there is no fundamental problem with doing so.

9.4 NN Architecture: a Layer-wise Organization

Neural Networks as neurons in graphs. Neural Networks are modeled as collections of neurons that are connected in an acyclic graph. In other words, the outputs of some neurons can become inputs to other neurons. Cycles are not allowed since that would imply an infinite loop in the forward pass of a network. Instead of an amorphous blobs of connected neurons, Neural Network models are often organized into distinct layers of neurons. For regular neural networks, the most common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections.

9.4.1 Naming Conventions

Notice that when we say N-layer neural network, we do not count the input layer. Therefore, a single-layer neural network describes a network with no hidden layers (input directly mapped to output). In that sense, you can sometimes hear people say that logistic regression or SVMs are simply a special case of single-layer Neural Networks. You may also hear these networks interchangeably referred to as "Artificial Neural Networks" (ANN) or "Multi-Layer Perceptrons" (MLP). Many people do not like the analogies between Neural Networks and real brains and prefer to refer to neurons as units.

9.4.2 Output Layer

Unlike all layers in a Neural Network, the output layer neurons most commonly do not have an activation function (or you can think of them as having a linear identity activation function). This is because the last output layer is usually taken to represent the class scores (e.g. in classification), which are arbitrary real-valued numbers, or some kind of real-valued target (e.g. in regression).

9.4.3 Sizing NN

The two metrics that people commonly use to measure the size of neural networks are the number of neurons, or more commonly the number of parameters. Here we propose two examples: The first network (left) has $4 + 2 = 6$ neurons (not counting the inputs), $[3x4] + [4x2] = 20$ weights and $4 + 2 = 6$ biases, for a total of 26 learnable parameters. The second network (right) has $4 + 4 + 1 = 9$ neurons, $[3x4] + [4x4] + [4x1] = 12 + 16 + 4 = 32$ weights and $4 + 4 + 1 = 9$ biases, for a total of 41 learnable parameters.

In general, modern Convolutional Networks contain on orders of 100 million parameters and are usually made up of approximately 10-20 layers (hence deep learning). However, as we will see the number of effective connections is significantly greater due to parameter sharing. More on this in the Convolutional Neural Networks module.

10 CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

So what does change? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

10.1 Architecture Overview

Recall: Regular Neural Nets. As we saw in the previous chapter, Neural Networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores.

Regular Neural Nets don't scale well to full images. In CIFAR-10, images are only of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have $32 * 32 * 3 = 3072$ weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have $200 * 200 * 3 = 120,000$ weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

3D volumes of neurons. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions $32 \times 32 \times 3$ (width, height, depth respectively). As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions 1x1x10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

10.2 Layers Used to Build CNN

As we described above, a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet architecture.

Example Architecture: Overview. We will go into more details below, but a simple ConvNet for CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. In more detail:

10.2.1 Input

INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.

10.2.2 Conv

CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.

10.2.3 Relu

RELU layer will apply an elementwise activation function, such as the $\max(0, x)$ thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).

10.2.4 Pool

POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].

10.2.5 FC

FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

In summary: A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores). There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular). Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function. Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't. Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't).

10.3 Convolutional Layer

The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting.

10.3.1 Overview and intuition without brain stuff

Lets first discuss what the CONV layer computes without brain/neuron analogies. The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

10.3.2 The brain view

If you're a fan of the brain/neuron analogies, every entry in the 3D output volume can also be interpreted as an output of a neuron that looks at only a small region in the input and shares parameters with all neurons to the left and right spatially (since these numbers all result from applying the same filter). We now discuss the details of the neuron connectivities, their arrangement in space, and their parameter sharing scheme.

10.3.3 Local Connectivity

When dealing with high-dimensional inputs such as images, as we saw above it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the filter size). The extent of the connectivity along the depth axis is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the entire depth of the input volume.

10.3.4 Spatial arrangement

We have explained the connectivity of each neuron in the Conv Layer to the input volume, but we haven't yet discussed how many neurons there are in the output volume or how they are arranged. Three hyperparameters control the size of the output volume: the depth, stride and zero-padding. We discuss these next:

1. First, the depth of the output volume is a hyperparameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a depth column (some people also prefer the term fibre).

2. Second, we must specify the stride with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.

3. As we will soon see, sometimes it will be convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes (most commonly as we'll see soon we will use it to exactly preserve the spatial size of the input volume so the input and output width and height are the same).

We can compute the spatial size of the output volume as a function of the input volume size (W), the receptive field size of the Conv Layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border. You can convince yourself that the correct formula for calculating how many neurons "fit" is given by $(W???F + 2P)/S + 1(W???F + 2P)/S + 1$. For example for a 7x7 input and a 3x3 filter with stride 1 and pad 0 we would get a 5x5 output. With stride 2 we would get a 3x3 output.

10.3.5 Constraints on strides

Note again that the spatial arrangement hyperparameters have mutual constraints. For example, when the input has size $W = 10$, no zero-padding is used $P = 0$, and the filter size is $F = 3$, then it would be impossible to use stride $S = 2$, since
$(W???F + 2P)/S + 1 = (10???3 + 0)/2 + 1 = 4.5(W???F + 2P)/S + 1 = (10???3 + 0)/2 + 1 = 4.5$, i.e. not an integer, indicating that the neurons don't "fit" neatly and symmetrically across the input. Therefore, this setting of the hyperparameters is considered to be invalid, and a ConvNet library could throw an exception or zero pad the rest to make it fit, or crop the input to make it fit, or something. As we will see in the ConvNet architectures section, sizing the ConvNets appropriately so that all the dimensions "work out" can be a real headache, which the use of zero-padding and some design guidelines will significantly alleviate.

10.3.6 Parameter Sharing

Parameter sharing scheme is used in Convolutional Layers to control the number of parameters. Using the real-world example above, we see that there are $55 * 55 * 96 = 290,400$ neurons in the first Conv Layer, and each has $11 * 11 * 3 = 363$ weights and 1 bias. Together, this adds up to $290400 * 364 = 105,705,600$ parameters on the first layer of the ConvNet alone. Clearly, this number is very high.

It turns out that we can dramatically reduce the number of parameters by making one reasonable assumption: That if one feature is useful to compute at some spatial position (x, y), then it should also be useful to compute at a different position (x_2, y_2). In other words, denoting a single 2-dimensional slice of depth as a depth slice (e.g. a volume of size [55x55x96] has 96 depth slices, each of size [55x55]), we are going to constrain the neurons in each depth slice to use the same weights and bias. With this parameter sharing scheme, the first Conv Layer in our example would now have only 96 unique set of weights (one for each depth slice), for a total of $96 * 11 * 11 * 3 = 34,848$ unique weights, or 34,944 parameters (+96 biases). Alternatively, all 55*55 neurons in each depth slice will now be using the same parameters. In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

Notice that if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can in each depth slice be computed as a convolution of the neuron's weights with the input volume (Hence the name: Convolutional Layer). This is why it is common to refer to the sets of weights as a filter (or a kernel), that is convolved with the input.

Note that sometimes the parameter sharing assumption may not make sense. This is especially the case when the input images to a ConvNet have some specific centered structure, where we should expect, for example, that completely different features should be learned on one side of the image than another. One practical example is when the input are faces that have been centered in the image. You might expect that different eye-specific or hair-specific features could (and should) be learned in different spatial locations. In that case it is common to relax the parameter sharing scheme, and instead simply call the layer a Locally-Connected Layer.

10.4 Implementation as Matrix Multiplication

Note that the convolution operation essentially performs dot products between the filters and local regions of the input. A common implementation pattern of the CONV layer is to take advantage of this fact and formulate the forward pass of a convolutional layer as one big matrix multiply as follows:

1. The local regions in the input image are stretched out into columns in an operation commonly called im2col. For example, if the input is [227x227x3] and it is to be convolved with 11x11x3 filters at stride 4, then we would take [11x11x3] blocks of pixels in the input and stretch each block into a column vector of size $11 * 11 * 3 = 363$. Iterating this process in the input at stride of 4 gives (227-11)/4+1 = 55 locations along both width and height, leading to an output matrix X_{col} of im2col of size [363 x 3025], where every column is a stretched out receptive field and there are $55 * 55 = 3025$ of them in total. Note that since the receptive fields overlap, every number in the input volume may be duplicated in multiple distinct columns.

2. The weights of the CONV layer are similarly stretched out into rows. For example, if there are 96 filters of size [11x11x3] this would give a matrix W_{row} of size [96 x 363].

3. The result of a convolution is now equivalent to performing one large matrix multiply np.dot($W_{\text{row}}, X_{\text{col}}$), which evaluates the dot product between every filter and every receptive field location. In our example, the output of this operation would be [96 x 3025], giving the output of the dot product of each filter at each location.

4. The result must finally be reshaped back to its proper output dimension [55x55x96].

11 DIMENSION REDUCTION

11.1 Bias-Variance Trade-off

As discussed earlier, there is a bias-variance trade-off. To do analyze this section, let us start with coefficients estimation. As usual, assume a model

$$y = f(z) + \epsilon,\ \epsilon \sim (0, \sigma^2)$$

In regression, our goal is to come up with some good regression function $\hat{f}(z) = z^T\hat{\beta}$. From Gausssian, Gauss-Markov, or machine learning techniques, we have different approaches for $\hat{\beta}^{ls}$. The question remains: can we do better?

Suppose we have an estimator $\hat{f}(z) = z^T\hat{\beta}$. To see if $\hat{f}(z) = z^T\hat{\beta}$ is a good candidate, we can ask ourselves two questions: (1) Is $\hat{\beta}$ close to the true β?, and (2) Will $\hat{f}(z)$ fit future observations well? To answer this, we consider mean squared error of our estimate $\hat{\beta}$:

$$\text{MSE}(\hat{\beta}) = \mathbb{E}[||\hat{\beta} - \beta||^2] = \mathbb{E}[(\hat{\beta} - \beta)^T(\hat{\beta} - \beta)]$$

To measure new measurements y_i' at the same z_i', we have

$$(z_1, y_1'), (z_2, y_2'), ..., (z_n, y_n')$$

and if our estimate (or fit) is a good model this estimate should also be close to new target y_j', which is the notion of prediction error. From decomposition, we have

$$\text{Error}(z_0) = \sigma_\epsilon^2 + \text{Bias}^2(\hat{f}(z_0)) + \text{Var}(\hat{f}(z_0))$$

Such a decomposition is known as the bias-variance tradeoff. As model becomes more complex (i.e. more terms included), local structure/curvature can be picked up. However, coefficient estiamtes suffer from high variance as more terms are included in the model. Hence, introducing a little bias in our estimate for β might lead to a substantial decrease in variance, and hence to a substantial decrease in prediction error.

11.2 PCR

Principal components analysis (PCA) is a popular approach for deriving a low-dimensional set of features from a large set of variables. PCA is a technique for reducing the dimension of a $n \times p$ data matrix X. The first principal component direction of the data is that along which the observations vary the most. There is also another interpretation for PCA: the first principal component vector defines the line that is as close as possible to the data.

11.2.1 The Principal Components Regression Approach

The principal components regression (PCR) approach involves constructing the first M principal components, $Z_1, ..., Z_m$, and then using these components as the predictors in a linear regression model that is fit using least squares. The key idea is that often a small number of principal components suffice to explain most of the variability in the data, as well as the relationship with the response. In other words, we assume that the directions in which $X_1, ..., X_p$ show the most variation are the directions that are associated with Y.

11.3 Step Variable Selection

A simple technique for selecting the most important variables is stepwise variable selection. The stepwise algorithm works by repeatedly adding or removing variables from the model, trying to "improve" the model at each step. When the algorithm can no longer improve the model by adding or subtracting variables, it stops and returns the new (and usually smaller) model.

Note that "improvement" does not just mean reducing the residual sum of squares (RSS) for the fitted model. Adding an additional variable to a model will not increase the RSS (see a statistics book for an explanation of why), but it does increase model complexity. Typically, AIC (Akaike's information criterion) is used to measure the value of each additional variable. The AIC is defined as AIC $=???2???\log(L) + k???cdf$, where L is the likelihood and edf is the equivalent degrees of freedom.

11.4 James-Stein

For $N \geq 3$, the James-Stein estimator everywhere dominates the MLE $\hat{\mu}^{(0)}$ in terms of expected total squared error; that is

$$E_\mu\{||\hat{\mu}^{(JS)} - \mu||^2\} < E_\mu\{||\hat{\mu}^{(MLE)} - \mu\}$$

for every choice of μ.

A quick proof of the theorem begins with the identity

$$(\hat{\mu}_i - \mu_i)^2 = (z_i - \hat{\mu}_i)^2 - (z_i - \mu_i)^2 + 2(\hat{\mu}_i - \mu_i)(z_i - \mu_i).$$

Summing the above equation over $i = 1, 2, ..., N$ and taking expectations gives

$$E_\mu\{||\hat{\mu} - \mu||^2\} = E\{||z - \hat{\mu}||^2\} - N + 2\sum_{i=1}^{N} \mathrm{cov}_\mu(\hat{\mu}_i, z_i),$$

where cov_μ indicates covariance under $z \sim \mathcal{N}_N(\mu, I)$. Integration by parts involving the multivariate normal density function $f_\mu(z) = (2\pi)^{-N/2}\exp\{-\frac{1}{2}\sum(z_i - \mu_i)^2\}$shows that

$$\mathrm{cov}_\mu(\hat{\mu}_i, z_i) = E_\mu\left\{\frac{\partial\hat{\mu}_i}{\partial z_i}\right\}.$$

Applying the simplified equation above to $\hat{\mu}^{(JS)} = (1 - \frac{N-2}{S})z$ gives

$$E_\mu\{||\hat{\mu}^{(JS)} - \mu||^2\} = N - E_\mu\left\{\frac{(N-2)^2}{S}\right\}$$

with $S = \sum z_i^2$ as before. The last term is positive if N exceeds 2, proving the theorem.

11.5 Ridge

11.5.1 Motivation

Stepwise variable selection simply fits a model using **lm()** function in R, but limits the number of variables in the model. In contrast, ridge regression places constraints on the size of the coefficients and fits a model using different computations. Ridge regression can be used to mitigate problems when there are several highly correlated variables in the underlying data. This condition (called multicollinearity) causes high variance in the results. Reducing the number, or impact, of regressors in the data can help reduce these problems.

We described how ordinary linear regression finds the coefficients that minimize the residual sum of squares. Ridge regression does something similar. Ridge regression attempts to minimize the sum of squared residuals plus a penalty for the coefficient sizes. The penalty is constant λ times the sum of squared coefficients. Specifically, ridge regression tries to minimize the following quantity:

$$\text{RSS}_{\text{ridge}}(c) = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{m} c_i^2$$

11.5.2 Ridge Approach

How does it work? Consider estimates for coefficients. If they are unconstrained, they can explode and are susceptible to high variance. To control variance, we might regularize the coefficients (how large they grow). We can impose ridge constraint:

$$\min \sum_{i=1}^{n}(y_i - \beta^T z_i)^2 \text{ such that } \sum_{j=1}^{p} \beta_j^2 \leq t$$

$$\Leftrightarrow \min (y - Z\beta)^T (y - Z\beta) \text{ such that } \sum_{j=1}^{p} \beta_j^2 \leq t$$

assuming that z is standardized with (mean 0 and unit variance) and y is centered. we can write the ridge constraint as the following penalized residual sum of squares (PRSS):

$$\begin{aligned} \text{PRSS}(\beta)_{l2} &= \textstyle\sum_{i=1}^{n}(y_i - z_i^T\beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \\ &= (y - Z\beta)^T(y - Z\beta) + \lambda ||\beta||_2^2 \end{aligned}$$

and the solution may have smaller average prediction error than least square estiamtes. Note that $\text{PRSS}(\beta)_{ls}$ is convex, and has a unique solution. Taking derivatives, we obtain

$$\frac{\partial \text{PRSS}(\beta)_{l2}}{\partial \beta} = -2Z^T(y - Z\beta) + 2\lambda\beta$$

and the solution to $\text{PRSS}(\hat{\beta})_{l2}$ is now seen to be

$$\hat{\beta}_{\lambda}^{\text{ridge}} = (Z^T Z + \lambda I_p)^{-1} Z^T y$$

Remember that in this case Z is standardized and y is centered. The solution is then indexed by the tuning parameter λ. For each λ, we have a solution. Hence, the λ's trace out a path of solutions (see Exercise 1. Other for graphical illustration). As a summary, λ is the shrinkage parameter. It controls the size of the coefficients and the amount of regularization. As λ tends to 0, we obtain the least squares solutions. Whilst λ tends to ∞, we have $\hat{\beta}_{\lambda=\infty}^{\text{ridge}} = 0$, which is an intercept-only model.

11.5.3 Proofs

What is left is tuning of the parameter λ. This is where ridge traces being introduced. Plot the components of $\hat{\beta}_\lambda^{\text{ridge}}$ against λ. Choose λ for which the coefficients are not rapidly changing and have "sensible signs".

First we prove that $\hat{\beta}_\lambda^{\text{ridge}}$ is biased. Let $R = Z^TZ$. Then consider

$$\begin{aligned}
\hat{\beta}_\lambda^{\text{ridge}} &= (Z^TZ + \lambda I_p)^{-1}Z^Ty \\
&= (R + \lambda I_p)^{-1}R(R^{-1}Z^Ty) \\
&= [R(I_p + \lambda^{-1})]^{-1}R[(Z^TZ)^{-1}Z^Ty] \\
&= (I_p + \lambda R^{-1})^{-1}R^{-1}R \\
&= (I_p + \lambda R^{-1})\hat{\beta}^{ls} \\
\Rightarrow & \\
\mathbb{E}(\hat{\beta}_\lambda^{\text{ridge}}) &= \mathbb{E}\{(I_p + \lambda R^{-1})\hat{\beta}^{ls}\} \\
&= (I_p + \lambda R^{-1})\beta \\
&\overset{\lambda \neq 0}{\neq} \beta
\end{aligned}$$

with this biased estimator, we rewrite l_2 PRSS as

$$\begin{aligned}
\text{PRSS}(\beta)_{l_2} &= \textstyle\sum_{i=1}^n (y_i - z_i^T\beta)^2 + \lambda \sum_{j=1}^p \beta_j^2 \\
&= \textstyle\sum_{i=1}^n (y_i - z_i^T\beta)^2 + \sum_{j=1}^p (0 - \sqrt{\lambda}\beta_j)^2
\end{aligned}$$

The l_2 criterion is the RSS for the augmented dataset:

$$Z_\lambda = \begin{pmatrix}
z_{1,1} & z_{1,2} & z_{1,3} & \dots & z_{1,p} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
z_{n,1} & z_{n,2} & z_{n,3} & \dots & z_{n,p} \\
\sqrt{\lambda} & 0 & 0 & \dots & 0 \\
0 & \sqrt{\lambda} & 0 & \dots & 0 \\
0 & 0 & \sqrt{\lambda} & \ddots & 0 \\
0 & 0 & 0 & \ddots & 0 \\
0 & 0 & 0 & \ddots & \sqrt{\lambda}
\end{pmatrix}; y_\lambda = \begin{pmatrix}
y_1 \\ \vdots \\ y_n \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0
\end{pmatrix}$$

and we can solve for

$$Z_\lambda = \begin{pmatrix} Z \\ \sqrt{\lambda}I_p \end{pmatrix}; y_\lambda = \begin{pmatrix} y \\ 0 \end{pmatrix}$$

Thus, the "least squares" solution for the augmented data is

$$\begin{aligned}
(Z_\lambda^TZ_\lambda)^{-1}Z_\lambda^Ty_\lambda &= \big((Z^T, \sqrt{\lambda}I_p)(\begin{pmatrix} Z \\ \sqrt{\lambda}I_p \end{pmatrix})\big)^{-1}(Z^T, \sqrt{\lambda}I_p)(\begin{pmatrix} y \\ 0 \end{pmatrix}) \\
&= (Z^TZ + \lambda I_p)^{-1}Z^Ty
\end{aligned}$$

□

11.5.4 Bayesian Framework

Suppose we imposed a multivariate Gaussian prior for β:

$$\beta \sim \mathcal{N}(0, \frac{1}{2p} I_p)$$

Then the posterior mean (and also posterior mode) of β is

$$\beta_\lambda^{\text{ridge}} = (Z^T Z + \lambda I_p)^{-1} Z^T y$$

The inverting of $Z^T Z$ can be computationally expensive. Instead, the singular value decomposition is utilized; that is,

$$Z = UDV^T,$$

where $U = (u_1, u_2, ..., u_p)$ is an $n \times p$ orthogonal matrix, $D = \text{diag}(d_1, d_2, ..., \geq d_p)$ is a $p \times p$ diagonal matrix consisting of the singular values $d_1 \geq d_2 \geq \ldots d_p \geq 0$, and $V^T = (v_1^T, v_2^T, ..., v_p^T)$ is a $p \times p$ matrix orthogonal matrix.

A consequence is that

$$\begin{array}{rcl} \hat{y}^{\text{ridge}} & = & Z\hat{\beta}_\lambda^{\text{ridge}} \\ & = & \sum_{j=1}^p \left(u_j \frac{d_j^2}{d_j^2 + \lambda} u_j^T\right) y \end{array}$$

Ridge regression has a relationship with principal components analysis (PCA). The fact is that the derived variable $\gamma_j = Zv_j = u_j d_j$ is the jth principal component (PC) of Z. Hence, ridge regression projects y onto these components with large d_j. Ridge regression shrinks the coefficients of low-variance components.

11.6 Lasso

Another technique for reducing the size of the coefficients (and thus reducing their impact on the final model) is the lasso. Like ridge regression, lasso regression puts a penalty on the size of the coefficients. However, the lasso algorithm uses a different penalty: instead of a sum of squared coefficients, the lasso sums the absolute value of the coefficients. (In math terms, ridge uses L2-norms, while lasso uses L1-norms.) Specifically, the lasso algorithm tries to minimize the following value:

$$\text{RSS}_{\text{lasso}}(c) = \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^m |c_i|$$

11.6.1 A Leading Example

Consider a linear regression, in which we observe N observations of an outcome variable y_i and p associated predictor variables (or features) $x_i = (x_{i1}, ..., x_{ip})^T$. The goal is to predict the outcome from the predictors, both for actual prediction with future data and also to discover which predictors play an important role. A linear regression model assumes that

$$y_i = \beta_0 + \sum_{j=1}^p x_{ij}\beta_j + e_i,$$

where β_0 and $\beta = (\beta_1, \beta_2, ..., \beta_p)$ are unknown parameters and e_i is an error term. The method of least squares provides estimates of the parameters by minimization of the least-squares objective function

$$\min_{\beta_0,\beta} \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2.$$

Typically all of the least-squares estimates from the above objective equation will be nonzero, which will make interpretation of the final model challenging if p is large.

Thus, there is a need to constrain or regularize the estimation process. In lasso or l_1-regularized regression, we estimate the parameters by solving the problem

$$\min_{\beta_0,\beta} \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \text{ subject to } ||\beta||_1 \leq t$$

where $||\beta||_1 = \sum_{j=1}^{p} |\beta_j|$ is the l_1 norm of β, and t is a user-specified parameter. We can think of t as a budget on the total l_1 norm of the parameter vector, and the lasso finds the best fit within this budge.

Why l_1 norm? It turns out that the l_1 norm is special. If the budget t is small enough, the lasso yields sparse solution vectors, having only some coordinates that are nonzero. This does not occur for p_q norms with $q > 1$; for $1 < 1$, the solutions are sparse but the problem is not convex and this makes the minimization very challenging computationally. The value $q = 1$ is the smallest value that yields a convex problem.

11.6.2 Lasso Estimator

Given a collection of N predictor-response pairs $\{(x_i, y_i)\}_{t=1}^{N}$, the lasso finds the solution $(\hat{\beta}_0, \hat{\beta})$ to the optimization problem

$$\min_{\beta_0,\beta}\left\{\frac{1}{2N}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2\right\}$$

$$\text{subject to } \sum_{j=1}^{p} |\beta_j| \leq t.$$

The constraint $\sum_{j=1}^{p} |\beta_j| \leq t$ can be written more compactly as the l_1-norm constraint $||\beta||_1 \leq t$. Furthermore, the above optimization equation is often represented using matrix vector notation. Let $\mathbf{y} = (y_1, ..., y_N)$ denote the N-vector of responses, and $\mathbf{X}$ be an $N \times p$ matrix with $x_i \in \mathbb{R}^p$ in the i^{th} row, then the optimization problem can be re-expressed as

$$\min_{\beta_0,\beta}\left\{\frac{1}{2N}||\mathbf{y} - \beta_0\mathbf{1} - \mathbf{X}\beta||_2^2\right\}$$

$$\text{subject to } ||\beta||_1 \leq t,$$

where $\mathbf{1}$ is the vector of N ones, and $|| \cdot ||_2$ denotes the usual Euclidean norm on vectors. The bound t is a kind of "budgeet": it limist the sum of the absolute vavlues of the parameter estimates. Since a shrunken parameter estimate corresponds to a more heavily-constrained model, this budge limits how well we can fit the data.

11.6.3 Compute Lasso Solution

First of all, the lasso problem is a convex program, specifically a quadratic program (QP) with a convex constraint. For convenience, we write the criterion in Lagrangian form:

$$\min_{\beta \in \mathbb{R}^p} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}.$$

First, let us consider a single predictor setting, based on samples $\{(z_i, y_i)\}_{i=1}^N$ (for convenience we have given the name z_i to this single x_{i1}). The problem then is to solve

$$\min_{\beta} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (y_i - z_i\beta)^2 + \lambda|\beta| \right\}$$

The standard approach is to use gradient descent with respect to β and set it to zero. However, the problem is that $|\beta|$ does not have a derivative at $\beta = 0$. However, direct inspection of the above objective function gives us

$$\hat{\beta} = \begin{cases} \frac{1}{N} < z, y > -\lambda & \text{if } \frac{1}{N} < z, y >> \lambda, \\ 0 & \text{if } \frac{1}{N} | < z, y > | \leq \lambda, \\ \frac{1}{N} < z, y > +\lambda & \text{if } \frac{1}{N} < z, y >< -\lambda \end{cases}$$

which we can write succinctly as

$$\hat{\beta} = S_\lambda(\frac{1}{N} < z, y >)$$

with soft-thresholding operator

$$S_\lambda(x) = \text{sign}(x)(|x| - \lambda)_+$$

translates its argument x toward zero by the amount λ and sets it to zero if $|x| \leq \lambda$.

Using the intuition from the univariate case, we can now develop a simple coordinatewise scheme for slving the predictors in some fixed (but arbitrary) order (say $j = 1, 2, ..., p$), where at the j^{th} step, we update the coefficient β_j by minimizing the objective function in this coordinate while holding fixed all other coefficients $\{\hat{\beta}_k, k \neq j\}$ at their current values. Hence, we write the objective as

$$\frac{1}{2N} \sum_{i=1}^{N} (y_i - \sum_{k \neq j} x_{ik}\beta_k - x_{ij}\beta_j)^2 + \lambda \sum_{k \neq j} |\beta_k| + \lambda|\beta_j|,$$

the solution for each β_j can be expressed $r_i^{(j)} = y_i - \sum_{k \neq j} x_{ik}\hat{\beta}_k$, which removes from the outcome the current fit from all but the j^{th} predictor. The j^{th} coefficient, in terms of partial residual, is updated as

$$\hat{\beta}_j = S_\lambda(\frac{1}{N} < x_j, r^{(j)} >),$$

where $r_i = y_i - \sum_{j=1}^{p} x_{ij}\hat{\beta}_j$ are the full residuals. The overall algorithm operates by applying this soft-thresholding update repeatedly in a cyclical manner, updating the coordinates of $\hat{\beta}$ (and hence the residual vectors) along the way.

11.7 Influence Measure: I Score

11.7.1 Background and Motivation

Professor Shaw-Hwa Lo proposed approaching prediction from a framework grounded in the theoretical correct prediction rate of a variable set as a parameter of interest. This framework allows us to define a measure of predictivity that enables assessing variable sets for, preferably high, predictivity. They first define the prediction rate for a variable set and consider, and ultimately reject, the naive estimator, a statistic based on the observed sample data, due to its inflated bias for moderate sample size and its sensitivity to noisy useless variables. We demonstrate that the I-score of the PR method of VS yields a relatively unbiased estimate of a parameter that is not sensitive to noisy variables and is a lower bound to the parameter of interest. Thus, the PR method using the II-score provides an effective approach to selecting highly predictive variables. We offer simulations and an application of the II-score on real data to demonstrate the statistic's predictive performance on sample data. We conjecture that using the partition retention and II-score can aid in finding variable sets with promising prediction rates; however, further research in the avenue of sample-based measures of predictivity is much desired.

The types of approaches and tools developed for feature selection are both diverse and varying in degrees of complexity. However, there is general agreement that three broad categories of feature selection methods exist: filter, wrapper, and embedded methods. Filter approaches tend to select variables through ranking them by various measures (correlation coefficients, entropy, information gains, chi-square, etc.). Wrapper methods use "black box" learning machines to ascertain the predictivity of groups of variables; because wrapper methods often involve retraining prediction models for different variable sets considered, they can be computationally intensive. Embedded techniques search for optimal sets of variables via a built-in classifier construction. A popular example of an embedded approach is the LASSO method for constructing a linear model, which penalizes the regression coefficients, shrinking many to zero. Often cross-validation is used to evaluate the prediction rates.

Often, though not always, the goal of these approaches is statistical inference. When this is the case, the researcher might be interested in understanding the mechanism relating the explanatory variables with a response. Although inference is clearly important, prediction is an important objective as well. In this case, the goal of these VS approaches is in inferring the membership of variables in the "important set." Various numerical criteria have been proposed to identify such variables [e.g., Akaike information criterion (AIC) and Bayesian information criterion (BIC), among others, which are associated with predictive performance under model assumptions made for the derivation of these criteria. However, these criteria were not designed to specifically correlate with predictivity. Indeed, we are unaware of a measure that directly attempts to evaluate a variable set's theoretical level of predictivity

An ideal measure for predictivity (or a good VSA measure) reflects a variable set's predictivity. In doing so, it would also guide VSA through screening out noisy variables and should correlate well with the out-of-sample correct prediction rate. We present a potential candidate measure, the II-score, for evaluating the predictivity of a given variable set in this section.

11.7.2 Theoretical Framework

11.7.2.1 Theorem

Under the assumptions that $\frac{n_d}{n} \to \lambda$, a value strictly between 0 and 1, and $\pi(d) = \pi(u) = 1/2$, then

$$\lim_{n\to\infty} \frac{s_n^2 I_{\Pi_{\mathbf{X}}}}{n} \overset{\mathcal{P}}{=} \lambda^2(1-\lambda)^2 \sum_{j\in\Pi_{\mathbf{X}}} [P(j|d) - P(j|u)]^2$$

where $\overset{\mathcal{P}}{=}$ indicates that the left-hand side converges in probability to the right-hand side and $s_n^2 = n_d n_u/n^2$.

Consider a set of n observations of disease phenotype Y (dichotomous or continuous) and a large number S of SNPs, X_1, X_2, $\ldots$, X_S. Randomly select a small group, m, of the SNPs. Following the same notation as in previous sections, we call this small group $\mathbf{X} = \{X_k, k = 1, ..., m\}$. Recall that X_k takes values 0, 1, and 2 (corresponding to three genotypes for a SNP locus: AA, A/B, and B/B). There are then $m_1 = 3^m$ possible values for $\mathbf{X}$'s. The n observations are partitioned into m_1 cells according to the values of the m SNPs (X_k's in $\mathbf{X}$), with n_j observations in the jth cell. We refer to this partition as $\Pi_{\mathbf{X}}$. The proposed I-score (denoted by $I_{\Pi_{\mathbf{X}}}$) is designed to place greater weight on cells that hold more observations:

$$I_{\Pi_{\mathbf{X}}} = \sum_{j=1}^{m_1} \cdot \frac{(\bar{Y}_j - \bar{Y})^2}{s_n^2/n_j} = \frac{\sum_{j=1}^{m_1} n_j^2 (\bar{Y}_j - \bar{Y})^2}{\sum_{i=1}^{n} (Y_i - \bar{Y})^2}$$

where $s_n^2 = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y})^2$. We note that the I-score is designed to capture the discrepancy between the conditional means of Y on $\{X_1, X_2, ..., X_m\}$ and the mean of Y.

In this section, we consider the special problem of a case-control experiment where there are n_d cases and n_U controls and the variable Y is 1 for a case and 0 for a control. Then $s_n^2 = (n_d n_u)/n^2$ where $n = n_d + n_u$.

We prove that the I-socre approaches a constant multiple of θ_I asymptotically.

Under the null hypothesis of no association between $\mathbf{X} = \{X_k, k = 1, ..., m\}$ and Y, $I_{\Pi_{\mathbf{X}}}$ can be asymptotically expressed as $\sum_{j=1}^{m_1} \lambda_j \chi_j^2$ (a weighted average), where λ_j is between 0 and 1 and $\sum_{j=1}^{m_1} \lambda_j$ is equal to $1 - \sum_{j=1}^{m_1} p_j^2$, where p_j is the cell j's probability. $\{\chi_j^2\}$ are m_1 chi-squares, each with degree of freedom, df $= 1$.

Moreover, the above formulation and properties of $I_{\Pi_{\mathbf{X}}}$ apply to the specified Y model with case-control study (where $Y = 1$ designates case and $Y = 0$ designates control). More specifically, in a case-control study with n_d cases and n_u controls (letting $n = n_d + n_u$), $ns_n^2 I_{\Pi_{\mathbf{X}}}$ can be expressed as the following:

$$\begin{aligned} ns_n^2 I_{\Pi_{\mathbf{X}}} &= \textstyle\sum_{j\in\Pi_{\mathbf{X}}} n_j^2 (\bar{Y}_j - \bar{Y})^2 \\ &= \textstyle\sum_{j\in\Pi_{\mathbf{X}}} (n_{d,j}^m + n_{u,j}^m)^2 \left(\frac{n_{d,j}^m}{n_{d,j}^m + n_{d,j}^m} - \frac{n_d}{n_d + n_u} \right) \\ &= \left(\frac{n_d n_u}{n_d + n_u} \right) \textstyle\sum_{j\in\Pi_{\mathbf{X}}} \left(\frac{n_{d,j}^m}{n_d} - \frac{n_{u,j}^m}{n_u} \right)^2 \end{aligned}$$

where $n_{d,j}^m$ and $n_{u,j}^m$ denote the numbers of cases and controls falling in jth cell, and $\Pi_{\mathbf{X}}$ stands for the partition formed by m variables in $\mathbf{X}$. Since the PR method seeks the partition that yields larger I-scores, one can decompose the following:

$$ns_n^2 I_{\Pi_{\mathbf{X}}} = \sum_{j\in\Pi_{\mathbf{X}}} n_j^2 (\bar{Y}_j - \bar{Y})^2 = A_n + B_n + C_n$$

where $A_n = \sum_{j\in\Pi_{\mathbf{X}}} n_j^2 (\bar{Y}_j - \mu_j)^2$, $B_n = \sum_{j\in\Pi_{\mathbf{X}}} n_j^2 (\bar{Y} - \mu_j)^2$, and $C_n = \sum_{j\in\Pi_{\mathbf{X}}} -2n_j^2 (\bar{Y}_j - \mu_j)(\bar{Y} - \mu_j)$. Here, μ_j and μ are the local and grand means of Y, that is, $E(\bar{Y}_j) = \mu_j$; $\bar{Y} = \mu = \frac{n_d}{n_d + n_u}$ for fixed n. It is easy to seethat both terms A_n and C_n, when divided by n^2 convere to 0 in probability as $n \to \infty$. We turn to the last term, B_n. Note that

$$\lim_n \frac{B_n}{n^2} \overset{\mathcal{P}}{=} \lim_n \sum_{j\in\Pi_{\mathbf{X}}} \left(\frac{n_j^2}{n^2} \right) (\mu_j - \mu)^2$$

In a case-control study, we have

$$\mu_j = \frac{n_d P(j|d)}{n_d P(j|d) + n_u P(j|u)}$$

and

$$\mu = \frac{n_d}{n_d + n_u}$$

Because for every j, $\frac{n_j}{n}$ converges (in probability) to $p_j = \lambda P(j|d) + (1-\lambda)P(j|u)$ as $n \to \infty$, if $\lim_n \frac{n_d}{n} = \lambda$, a fixed a constant between 0 and 1, it follows that

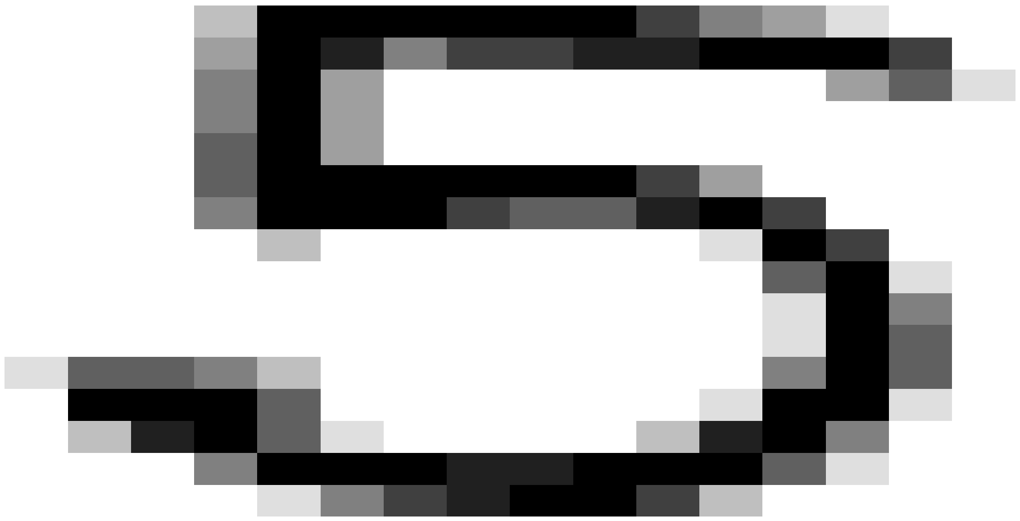
$$\begin{aligned}
\tfrac{B_n}{n^2} &= \textstyle\sum_{j\in\Pi_{\mathbf{X}}} \left(\frac{n_j^2}{n^2}\right)(\mu_j - \mu)^2 \\
&\xrightarrow{\mathcal{P}} \textstyle\sum_{j\in\Pi_{\mathbf{X}}} p_j^2 \left(\frac{\lambda P(j|d)}{\lambda P(j|d)+(1-\lambda)P(j|u)}\right)^2 \text{ as } n \to \infty \\
&= \textstyle\sum_{j\in\Pi_{\mathbf{X}}} \{\lambda P(j|d) - \lambda[\lambda P(j|d) + (1-\lambda)P(j|u)]\}^2 \\
&= \textstyle\sum_{j\in\Pi_{\mathbf{X}}} \{\lambda(1-\lambda)P(j|d) - [\lambda(1-\lambda)P(j|u)]\}^2 \\
&= \lambda^2(1-\lambda)^2 \textstyle\sum_{j\in\Pi_{\mathbf{X}}} [P(j|d) - P(j|u)]^2
\end{aligned}$$

Thus, neglecting the constant term in the above equation, the I-score can guide a search for X partitions, which will lead to finding larger values of the summation term $\sum_{j\in\Pi_{\mathbf{X}}} [P(j|d) - P(j|u)]^2$.

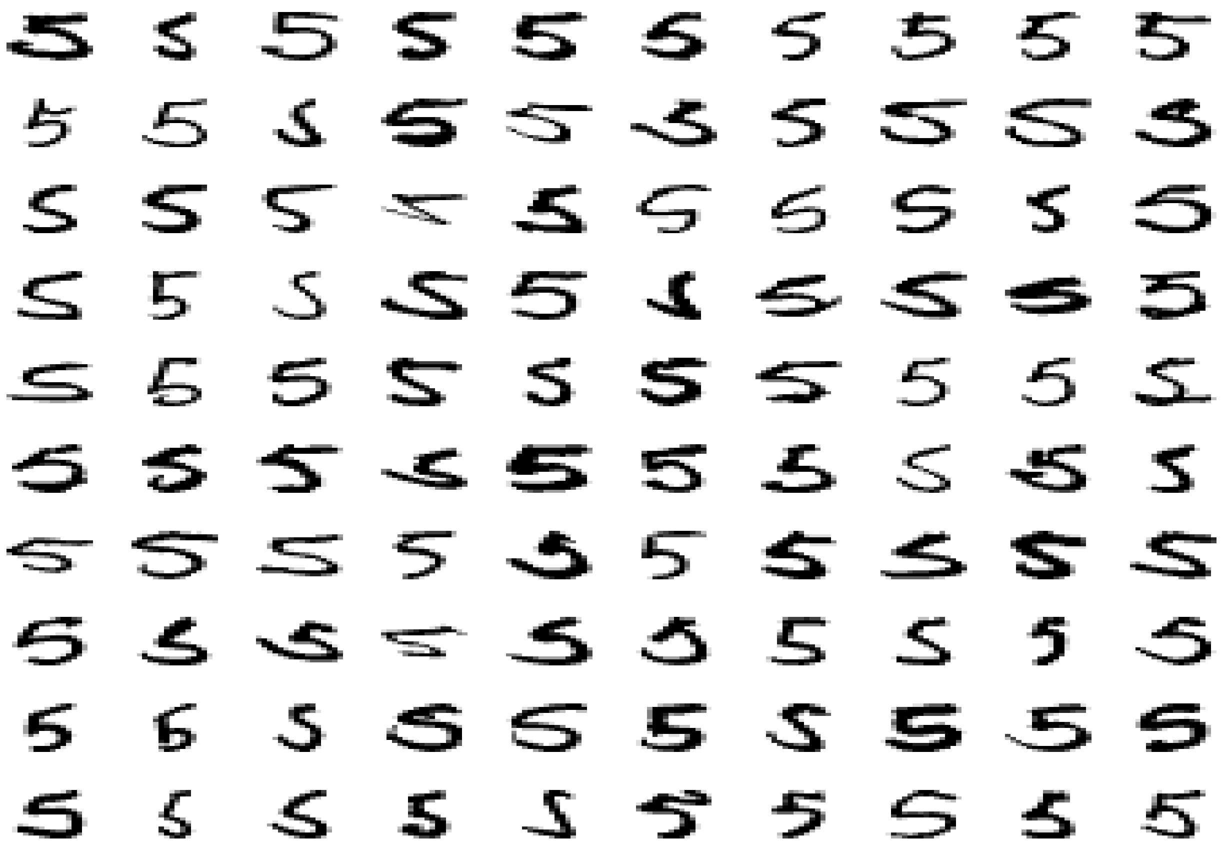
12 Exercise 1

Consider the famous example of handwritten digits recognition. The exercise we can write functions of visualization of

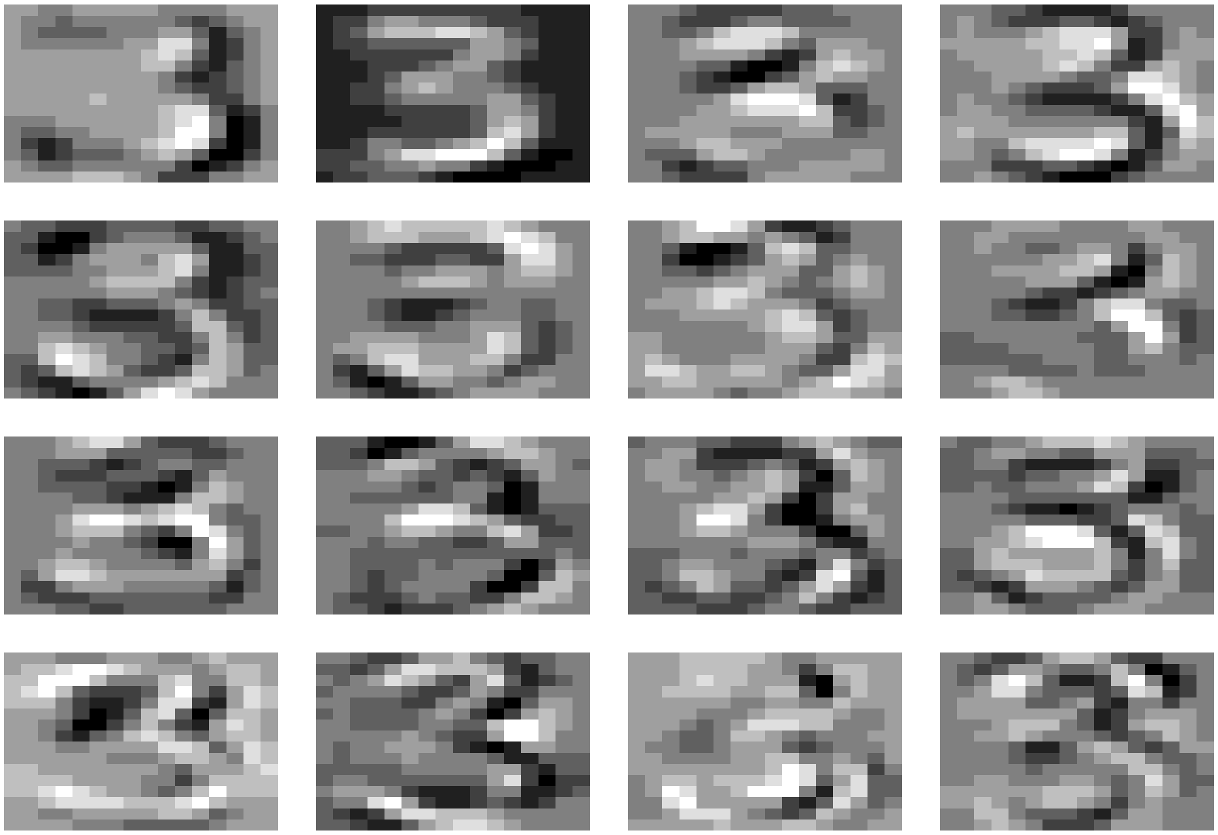
```r
# Setup dataset:
zip<-read.table("zip.train", header=FALSE, sep=" ")
zip<-zip[, -258]
x_train<-as.matrix(zip[,2:257])
y_train<-as.matrix(zip[,1])
#write.csv(zip, file="zip_train.csv", row.names = FALSE, col.names = FALSE)

test<-read.table("ziptest.txt", header = FALSE, sep=" ")
x_test<-as.matrix(test[, 2:257])
y_test<-as.matrix(test[, 1])
y_test<-as.factor(y_test)

zip.3<-read.table("train.3.txt", header=FALSE, sep=",")
zip.5<-read.table("train.5.txt", header=FALSE, sep=",")
zip.3<-as.matrix(zip.3)
n.3<-length(zip.3[,1])
zip.5<-as.matrix(zip.5)
n.5<-length(zip.5[,1])
data<-rbind(zip.3, zip.5)

### Write a function of data visualization, input is a vector of length 256 ###
output.image<-function(vector) {
    digit<-matrix(vector, nrow=16, ncol=16)
    index= seq(from=16, to =1, by=-1)
    sym_digit = digit[,index]
    image(sym_digit, col= gray((8:0)/8), axes=FALSE)
    #image(digit, col= gray((8:0)/8), axes=FALSE)
}

# Ex:
output.image(zip.5[3,])
```



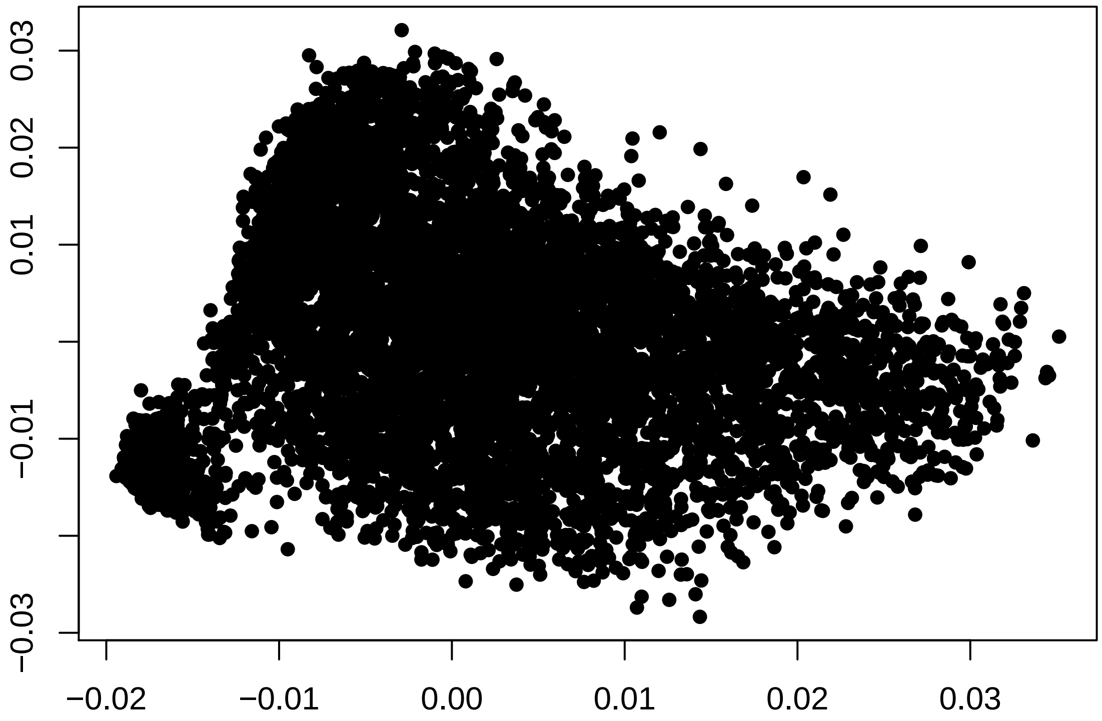
```
# Comment: The output of the function, output.image(), will give us
# a 16x16 pixel image. In this case, the data set zip.5 is examples
# of observations of digit 5.

### Visualization of data ###
par(mfrow=c(10,10),mai=c(0.1,0.1,0.1,0.1))
for(i in 1:100) {
  output.image(zip.5[i,])
    #output.image(zip.3[i,])
}
```



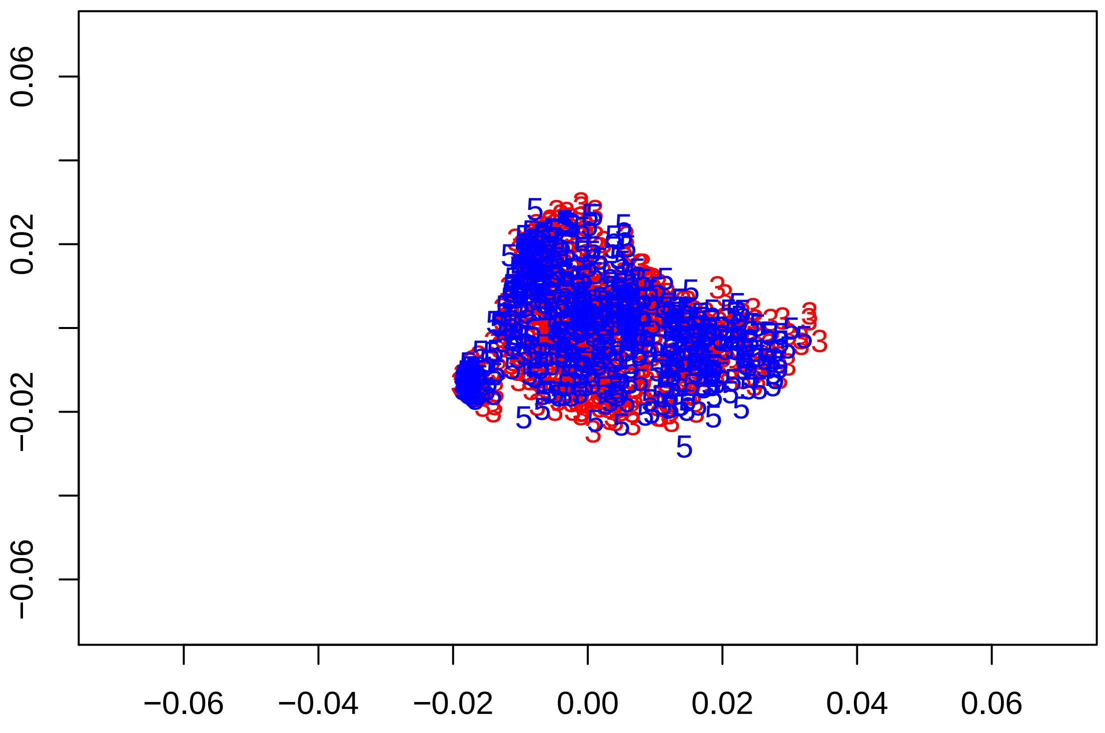
```
# Comment: This is a collection of visualization of
# the first 100 observations of digit 5 from the same datset
# as the above.

### Center the data ###
scaled.3<-scale(zip.3,center=TRUE, scale=FALSE)
scaled.data<-scale(data, center=TRUE, scale=FALSE)
x_train_scaled<-scale(x_train, center=TRUE, scale = FALSE)

pca<-svd(scaled.3)
par(mfrow=c(4,4), mai=c(0.1,0.1, 0.1, 0.1))
for(j in 1:16) {
  output.image(pca$v[,j])
}
```



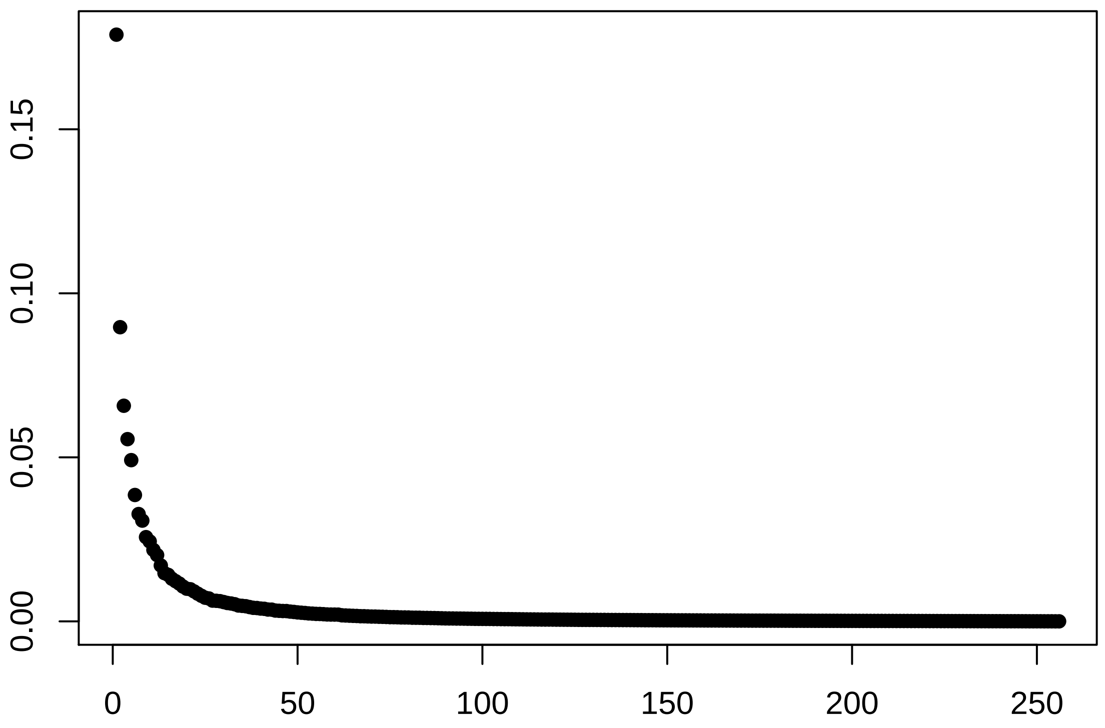
```
# Comment: for different dimensions we observe the input image
# is transformed into different images as presented in the graph.

# Principal Component Analysis:
pca<-svd(scaled.data)
pca<-svd(scaled.3)
pca<-svd(x_train_scaled)

### Projections on the subspace spanned by the first two principle components ###
par(mfrow=c(1,1), mai=c(0.6, 0.6, 0.6, 0.6))
plot(pca$u[,1], pca$u[, 2], pch=16,
     xlab="First Principle Component",
     ylab="Second Principle Component" )
```



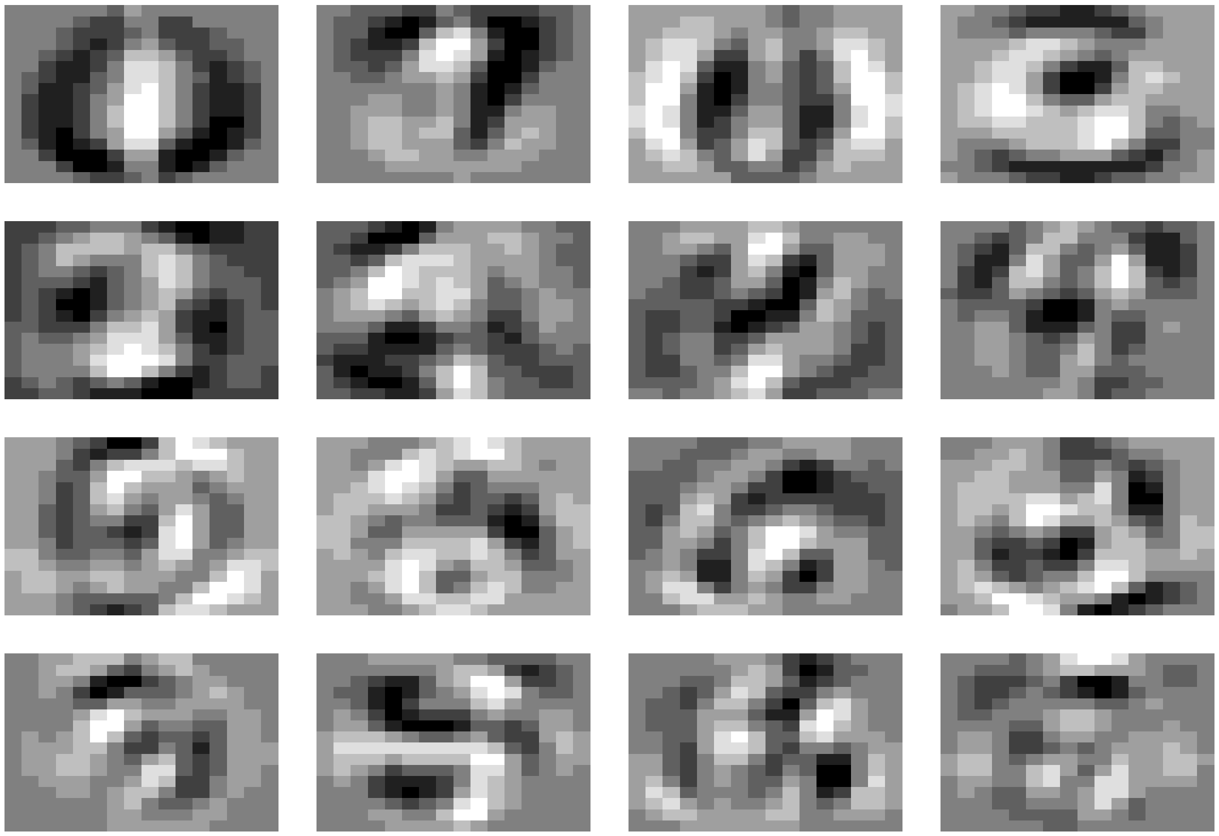
```
plot(pca$u[1:n.3, 1], pca$u[1:n.3, 2], pch="3", col="red",
     xlim=c(-0.07, 0.07), ylim=c(-0.07, 0.07),
     xlab="First Principle Component",
     ylab="Second Principle Component")
points(pca$u[(n.3+1):(n.3+n.5), 1], pca$u[(n.3+1):(n.3+n.5), 2], pch="5", col="blue")
```



```
### Scree plot ###
plot(seq(from=1,to=256, by=1), (pca$d)^2/sum((pca$d)^2),
     xlab="Priciple componnets",
     ylab="Proportion of variance explained",
     pch=16)
```



```
### Visualization of principel components ###
par(mfrow=c(4,4),mai=c(0.1,0.1,0.1,0.1))
for(j in 1:16) {
    output.image(pca$v[,j])
}
```

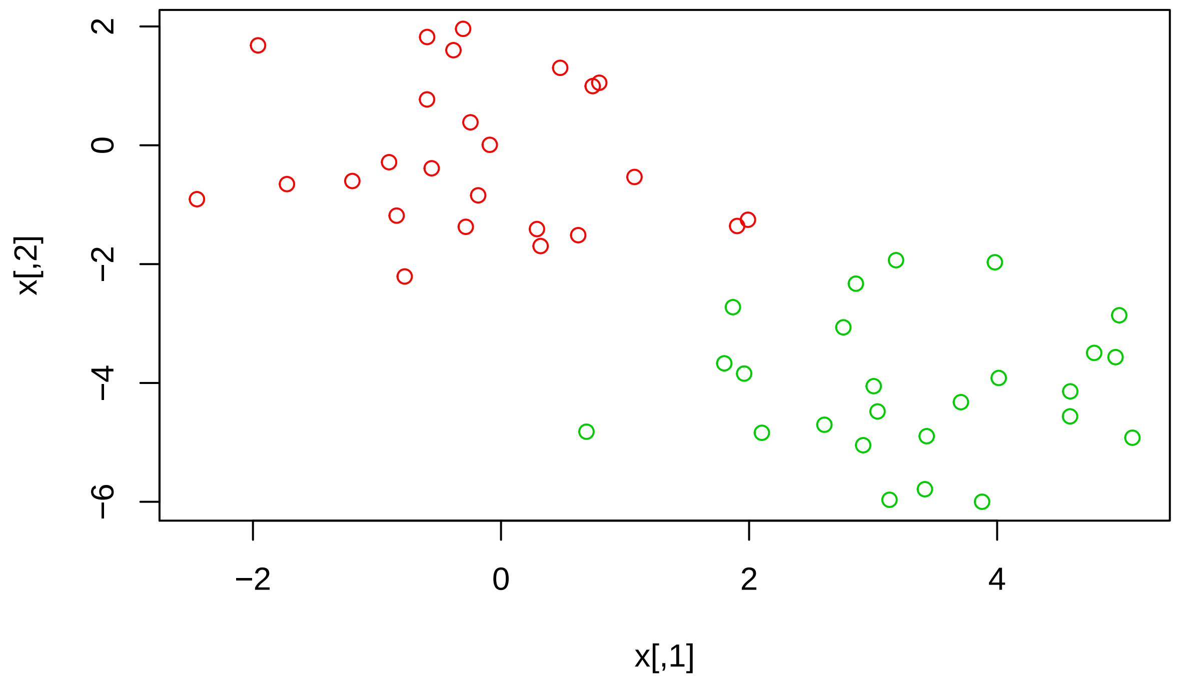


12.1 K-Means

```
set.seed(2)
x <- matrix(rnorm(50*2), ncol = 2)
x[1:25, 1] <- x[1:25, 1]+3
x[1:25, 2] <- x[1:25, 2]-4
km.out <- kmeans(x,2,nstart = 20)
km.out$cluster
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
plot(x, col=(km.out$cluster + 1))
```



```
# Comment: this will give you a plot.

set.seed(3)
km.out = kmeans(x,3,nstart=2)
km.out$tot.withinss
```

```
## [1] 97.97927
```

```
km.out = kmeans(x,3,nstart=2000)
km.out$tot.withinss
```

```
## [1] 97.97927
```

12.2 Linear Regression

The linear models always take the form $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \epsilon$. We can maximize $beta_p$ by minimizing RSS. We can start with a linear model. We adjust or we can make predictions. For predictions made, we can select a model that we beleive ideal or we can make changes to this model.

```
# Remember to install packages first:
library(MASS)
library(ISLR)

# Linear Model
lm.fit <- lm(medv ~ lstat, data=Boston)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.168  -3.990  -1.318   2.034  24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384    0.56263   61.41   <2e-16 ***
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

```
# Comment: we can summarize results by summary().
# We read the results and observe estimate, std. error, and
# t-value. We can check whether each predictor is significant
# or not. We can also check p-value. In this case,
# both parameters are important and we need to retain them.

# Multi-various Linear Model
lm.fit <- lm(medv ~ lstat + age, data = Boston)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.981  -3.978  -1.283   1.968  23.158
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.22276    0.73085  45.458  < 2e-16 ***
## lstat       -1.03207    0.04819 -21.416  < 2e-16 ***
## age          0.03454    0.01223   2.826  0.00491 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.173 on 503 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495
## F-statistic:   309 on 2 and 503 DF,  p-value: < 2.2e-16
```

```
lm.fit <- lm(medv ~ ., data = Boston)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ ., data = Boston)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595  -2.730  -0.518   1.777  26.199
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
## chas         2.687e+00  8.616e-01   3.118 0.001925 **
## nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
## rm           3.810e+00  4.179e-01   9.116  < 2e-16 ***
## age          6.922e-04  1.321e-02   0.052 0.958229
## dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
## tax         -1.233e-02  3.760e-03  -3.280 0.001112 **
## ptratio     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
## black        9.312e-03  2.686e-03   3.467 0.000573 ***
## lstat       -5.248e-01  5.072e-02 -10.347  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

```r
# Comment: for eaxmple, we can observe the p-value
# for age is very large. The first step we can
# simply drop the variable age.

# Interaction term:
summary(lm(medv~lstat*age,data=Boston))
```

```
##
## Call:
## lm(formula = medv ~ lstat * age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.806  -4.045  -1.333   2.085  27.552
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 36.0885359  1.4698355  24.553  < 2e-16 ***
## lstat       -1.3921168  0.1674555  -8.313 8.78e-16 ***
## age         -0.0007209  0.0198792  -0.036   0.9711
## lstat:age    0.0041560  0.0018518   2.244   0.0252 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.149 on 502 degrees of freedom
## Multiple R-squared:  0.5557, Adjusted R-squared:  0.5531
## F-statistic: 209.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

```r
summary(lm(medv~lstat:age,data=Boston))
```

```
##
## Call:
## lm(formula = medv ~ lstat:age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -13.347  -4.372  -1.534   1.914  27.193
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.1588631  0.4828240   62.46   <2e-16 ***
## lstat:age   -0.0077146  0.0003799  -20.31   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.827 on 504 degrees of freedom
## Multiple R-squared:  0.4501, Adjusted R-squared:  0.449
## F-statistic: 412.4 on 1 and 504 DF,  p-value: < 2.2e-16
```

```r
# Higher degree:
lm.fit2 <- lm(medv ~ lstat + lstat^2, data=Boston); summary(lm.fit2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + lstat^2, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.168  -3.990  -1.318   2.034  24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384    0.56263   61.41   <2e-16 ***
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

```r
# Notice that the square is not working; we need to use
# function I() to make sure the new variable is calculted
# properly.
lm.fit2 <- lm(medv ~ lstat + I(lstat^2), data=Boston); summary(lm.fit2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2834  -3.8313  -0.5295   2.3095  25.4148
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.862007   0.872084   49.15   <2e-16 ***
## lstat       -2.332821   0.123803  -18.84   <2e-16 ***
## I(lstat^2)   0.043547   0.003745   11.63   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.524 on 503 degrees of freedom
## Multiple R-squared:  0.6407, Adjusted R-squared:  0.6393
## F-statistic: 448.5 on 2 and 503 DF,  p-value: < 2.2e-16
```

```
lm.fit <- lm(medv ~ lstat, data=Boston)
anova(lm.fit, lm.fit2)
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ lstat
## Model 2: medv ~ lstat + I(lstat^2)
##   Res.Df   RSS Df Sum of Sq     F    Pr(>F)
## 1    504 19472
## 2    503 15347  1    4125.1 135.2 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
lm.fit1 <- lm(medv ~ .-age-indus, data=Boston)
lm.fit2 <- lm(medv ~., data = Boston)
anova(lm.fit1, lm.fit2)
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ (crim + zn + indus + chas + nox + rm + age + dis + rad +
##     tax + ptratio + black + lstat) - age - indus
## Model 2: medv ~ crim + zn + indus + chas + nox + rm + age + dis + rad +
##     tax + ptratio + black + lstat
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1    494 11081
## 2    492 11079  2    2.5794 0.0573 0.9443
```

```
# Comment:
# This way we can detect a better model without the variables
# age and indus.

# Use a different data
head(Carseats, 3) # Quick view
```

```
##   Sales CompPrice Income Advertising Population Price ShelveLoc Age
## 1  9.50       138     73          11        276   120       Bad  42
## 2 11.22       111     48          16        260    83      Good  65
## 3 10.06       113     35          10        269    80    Medium  59
##   Education Urban  US
## 1        17   Yes Yes
## 2        10   Yes Yes
## 3        12   Yes Yes
```

```r
summary(lm(Sales ~., data=Carseats))
```

```
##
## Call:
## lm(formula = Sales ~ ., data = Carseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8692 -0.6908  0.0211  0.6636  3.4115
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      5.6606231  0.6034487   9.380  < 2e-16 ***
## CompPrice        0.0928153  0.0041477  22.378  < 2e-16 ***
## Income           0.0158028  0.0018451   8.565 2.58e-16 ***
## Advertising      0.1230951  0.0111237  11.066  < 2e-16 ***
## Population       0.0002079  0.0003705   0.561    0.575
## Price           -0.0953579  0.0026711 -35.700  < 2e-16 ***
## ShelveLocGood    4.8501827  0.1531100  31.678  < 2e-16 ***
## ShelveLocMedium  1.9567148  0.1261056  15.516  < 2e-16 ***
## Age             -0.0460452  0.0031817 -14.472  < 2e-16 ***
## Education       -0.0211018  0.0197205  -1.070    0.285
## UrbanYes         0.1228864  0.1129761   1.088    0.277
## USYes           -0.1840928  0.1498423  -1.229    0.220
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.019 on 388 degrees of freedom
## Multiple R-squared:  0.8734, Adjusted R-squared:  0.8698
## F-statistic: 243.4 on 11 and 388 DF,  p-value: < 2.2e-16
```

12.3 Logistic Regression

```r
# Use Logistic Regression
head(Smarket, 3) # Quick view
```

```
##   Year  Lag1   Lag2   Lag3   Lag4   Lag5 Volume  Today Direction
## 1 2001 0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959        Up
## 2 2001 0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032        Up
## 3 2001 1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623      Down
```

```r
glm.fit <- glm(Direction ~.-Today-Year,
               data = Smarket, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ . - Today - Year, family = binomial,
##     data = Smarket)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.446  -1.203   1.065   1.145   1.326
```

```
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000   0.240736  -0.523    0.601
## Lag1        -0.073074   0.050167  -1.457    0.145
## Lag2        -0.042301   0.050086  -0.845    0.398
## Lag3         0.011085   0.049939   0.222    0.824
## Lag4         0.009359   0.049974   0.187    0.851
## Lag5         0.010313   0.049511   0.208    0.835
## Volume       0.135441   0.158360   0.855    0.392
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.6  on 1243  degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3
```

```
glm.probs <- predict(glm.fit, type = "response")
glm.pred = rep("Down", 1250)
glm.pred[glm.probs > .5] = "Up"
table(glm.pred, Smarket$Direction)
```

```
##
## glm.pred Down  Up
##     Down  145 141
##     Up    457 507
```

```
sum(diag(table(glm.pred, Smarket$Direction)))/sum(table(glm.pred, Smarket$Direction))
```

```
## [1] 0.5216
```

12.4 LDA

LDA can assist us dealing with data

```
head(Smarket, 3) # Quick view
```

```
##   Year  Lag1   Lag2   Lag3   Lag4   Lag5 Volume  Today Direction
## 1 2001 0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959        Up
## 2 2001 0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032        Up
## 3 2001 1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623      Down
```

```
lda.fit <- lda(Direction ~ Lag1 + Lag2, data = Smarket)
lda.pred <- predict(lda.fit, Smarket)
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

```
table(lda.pred$class, Smarket$Direction)
```

```
##
##        Down  Up
##   Down  114 102
##   Up    488 546
```

```r
sum(diag(table(lda.pred$class, Smarket$Direction)))/sum(table(lda.pred$class, Smarket$Direction))
```

```
## [1] 0.528
```

12.5 PCA

```r
# Principal Components Analysis
states=row.names(USArrests)
states
```

```
##  [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"
##  [5] "California"     "Colorado"       "Connecticut"    "Delaware"
##  [9] "Florida"        "Georgia"        "Hawaii"         "Idaho"
## [13] "Illinois"       "Indiana"        "Iowa"           "Kansas"
## [17] "Kentucky"       "Louisiana"      "Maine"          "Maryland"
## [21] "Massachusetts"  "Michigan"       "Minnesota"      "Mississippi"
## [25] "Missouri"       "Montana"        "Nebraska"       "Nevada"
## [29] "New Hampshire"  "New Jersey"     "New Mexico"     "New York"
## [33] "North Carolina" "North Dakota"   "Ohio"           "Oklahoma"
## [37] "Oregon"         "Pennsylvania"   "Rhode Island"   "South Carolina"
## [41] "South Dakota"   "Tennessee"      "Texas"          "Utah"
## [45] "Vermont"        "Virginia"       "Washington"     "West Virginia"
## [49] "Wisconsin"      "Wyoming"
```

```r
names(USArrests)
```

```
## [1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

```r
apply(USArrests, 2, mean)
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```r
apply(USArrests, 2, var)
```

```
##     Murder    Assault   UrbanPop       Rape
##   18.97047 6945.16571  209.51878   87.72916
```

```r
pr.out=prcomp(USArrests, scale=TRUE)
names(pr.out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

```r
pr.out$center
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```r
pr.out$scale
```

```
##    Murder   Assault  UrbanPop      Rape
##  4.355510 83.337661 14.474763  9.366385
```

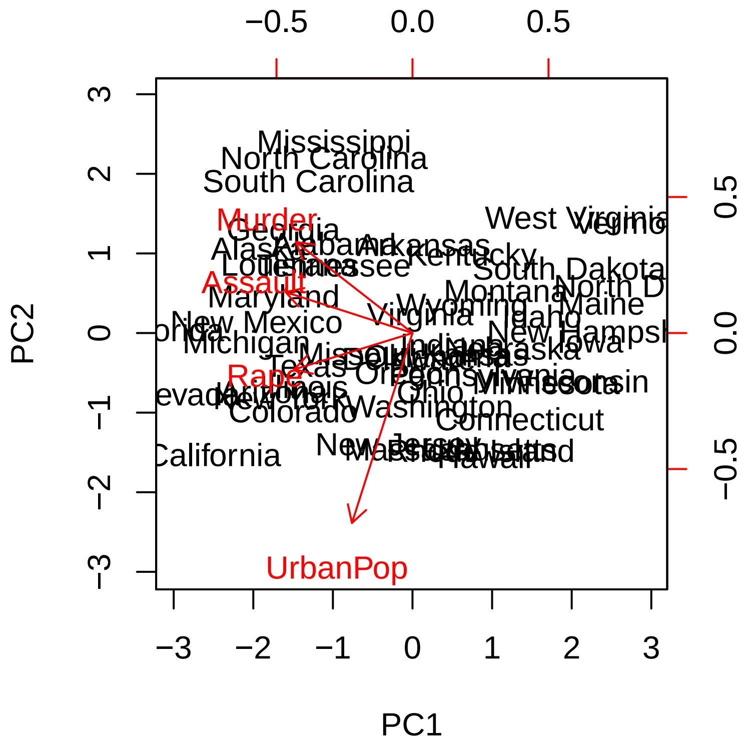
```r
pr.out$rotation
```

```
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

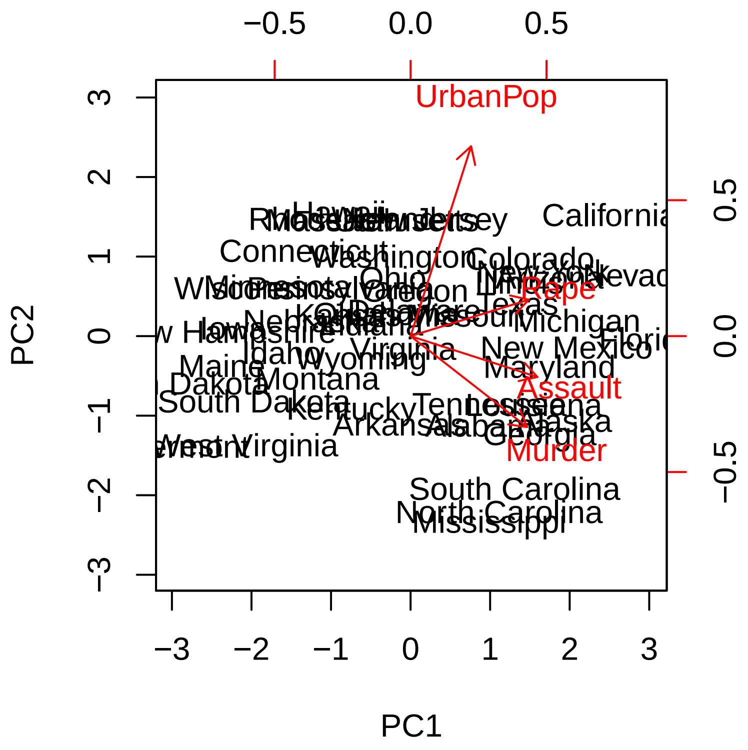
```
dim(pr.out$x)
```

```
## [1] 50  4
```

```
biplot(pr.out, scale=0)
```



```
pr.out$rotation=-pr.out$rotation
pr.out$x=-pr.out$x
biplot(pr.out, scale=0)
```



```r
pr.out$sdev
```

```
## [1] 1.5748783 0.9948694 0.5971291 0.4164494
```

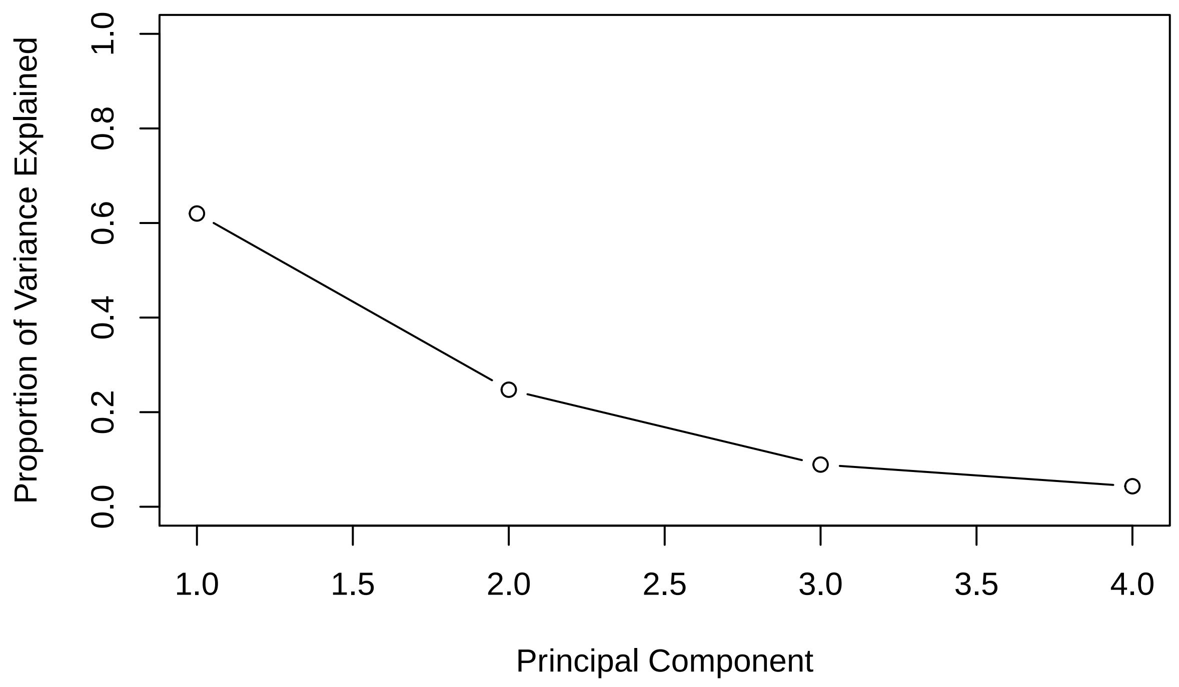
```r
pr.var=pr.out$sdev^2
pr.var
```

```
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
```

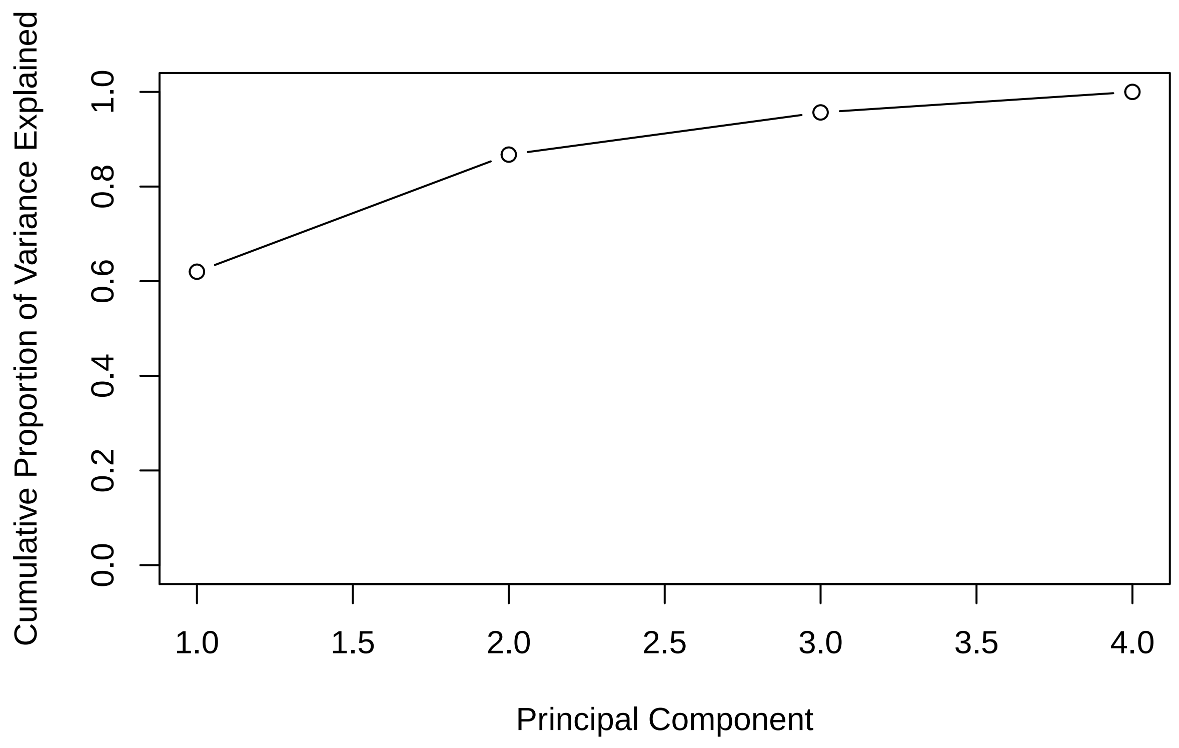
```r
pve=pr.var/sum(pr.var)
pve
```

```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

```r
plot(pve, xlab="Principal Component",
     ylab="Proportion of Variance Explained",
     ylim=c(0,1),type='b')
```



```
plot(cumsum(pve),
     xlab="Principal Component",
     ylab="Cumulative Proportion of Variance Explained",
     ylim=c(0,1),type='b')
```



```
a=c(1,2,8,-3)
cumsum(a)
```

```
## [1]  1  3 11  8
```

12.6 Application: Stock Data; Logistic, LDA, QDA, and KNN

```
# The Stock Market Data
library(ISLR)
names(Smarket)
```

```
## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"
```

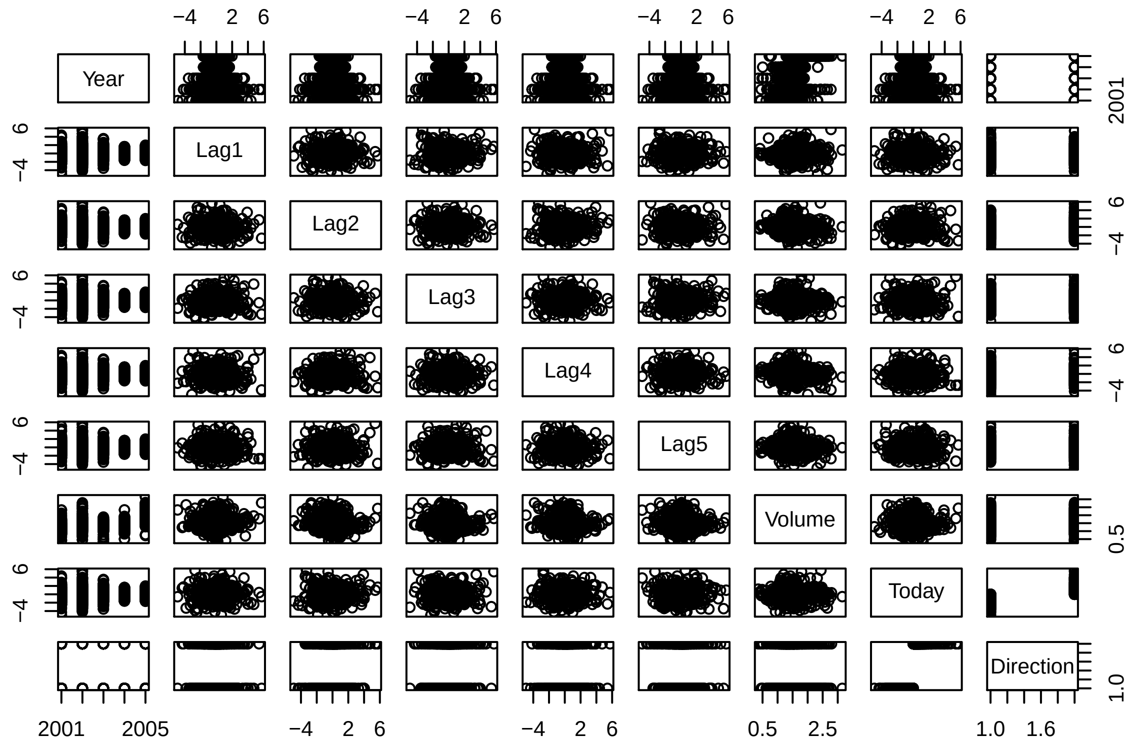
```
dim(Smarket)
```

```
## [1] 1250    9
```

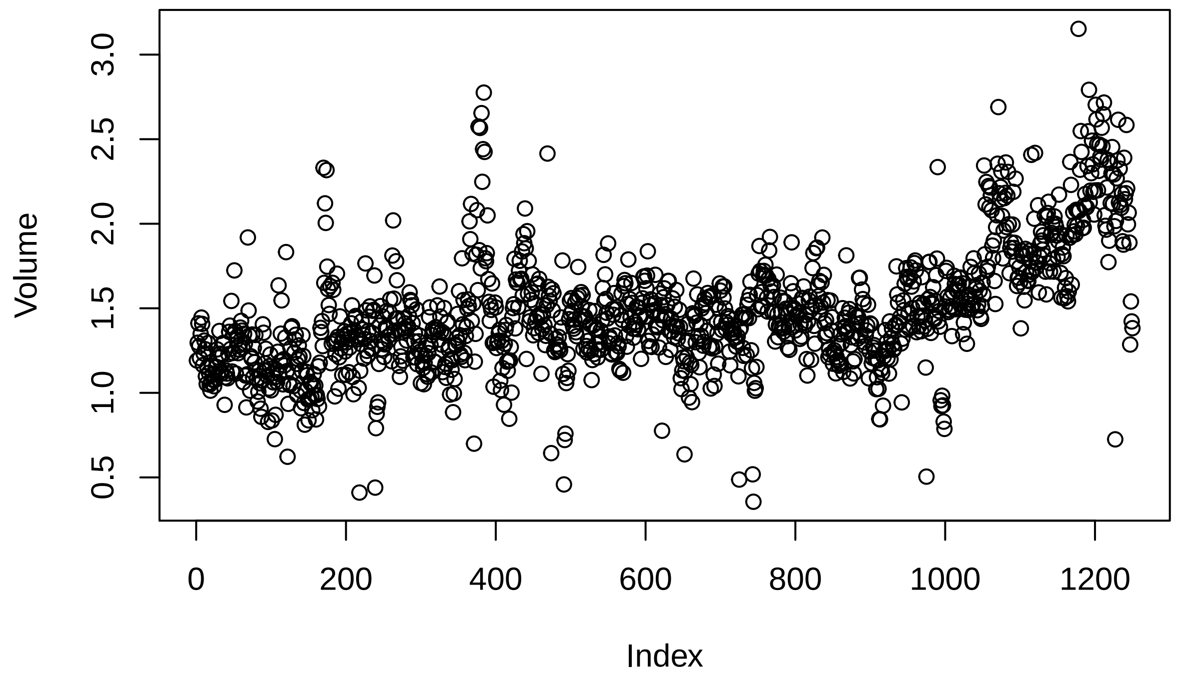
```
summary(Smarket)
```

```
##       Year           Lag1                Lag2
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000
##  1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500
##  Median :2003   Median : 0.039000   Median : 0.039000
##  Mean   :2003   Mean   : 0.003834   Mean   : 0.003919
##  3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750
##  Max.   :2005   Max.   : 5.733000   Max.   : 5.733000
##        Lag3                Lag4                Lag5
##  Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.92200
##  1st Qu.:-0.640000   1st Qu.:-0.640000   1st Qu.:-0.64000
##  Median : 0.038500   Median : 0.038500   Median : 0.03850
##  Mean   : 0.001716   Mean   : 0.001636   Mean   : 0.00561
##  3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.59700
##  Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.73300
##      Volume           Today            Direction
##  Min.   :0.3561   Min.   :-4.922000   Down:602
##  1st Qu.:1.2574   1st Qu.:-0.639500   Up  :648
##  Median :1.4229   Median : 0.038500
##  Mean   :1.4783   Mean   : 0.003138
##  3rd Qu.:1.6417   3rd Qu.: 0.596750
##  Max.   :3.1525   Max.   : 5.733000
```

```r
pairs(Smarket)
```



```r
#cor(Smarket)
#cor(Smarket[,-9])
attach(Smarket)
plot(Volume)
```



```r
# Logistic Regression

glm.fit=glm(
  Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
  data=Smarket,family=binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Smarket)
##
## Deviance Residuals:
##    Min       1Q   Median       3Q      Max
## -1.446   -1.203    1.065    1.145    1.326
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000   0.240736  -0.523    0.601
## Lag1        -0.073074   0.050167  -1.457    0.145
## Lag2        -0.042301   0.050086  -0.845    0.398
## Lag3         0.011085   0.049939   0.222    0.824
## Lag4         0.009359   0.049974   0.187    0.851
## Lag5         0.010313   0.049511   0.208    0.835
## Volume       0.135441   0.158360   0.855    0.392
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.6  on 1243  degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3
```

```
coef(glm.fit)
```

```
##  (Intercept)         Lag1         Lag2         Lag3         Lag4
## -0.126000257 -0.073073746 -0.042301344  0.011085108  0.009358938
##         Lag5       Volume
##  0.010313068  0.135440659
```

```
summary(glm.fit)$coef
```

```
##                 Estimate Std. Error    z value  Pr(>|z|)
## (Intercept) -0.126000257 0.24073574 -0.5233966 0.6006983
## Lag1        -0.073073746 0.05016739 -1.4565986 0.1452272
## Lag2        -0.042301344 0.05008605 -0.8445733 0.3983491
## Lag3         0.011085108 0.04993854  0.2219750 0.8243333
## Lag4         0.009358938 0.04997413  0.1872757 0.8514445
## Lag5         0.010313068 0.04951146  0.2082966 0.8349974
## Volume       0.135440659 0.15835970  0.8552723 0.3924004
```

```
summary(glm.fit)$coef[,4]
```

```
## (Intercept)        Lag1        Lag2        Lag3        Lag4        Lag5
##   0.6006983   0.1452272   0.3983491   0.8243333   0.8514445   0.8349974
##      Volume
##   0.3924004
```

```
glm.probs=predict(glm.fit,type="response")
glm.probs[1:10]
```

```
##         1         2         3         4         5         6         7
## 0.5070841 0.4814679 0.4811388 0.5152224 0.5107812 0.5069565 0.4926509
##         8         9        10
## 0.5092292 0.5176135 0.4888378
```

```
contrasts(Direction)
```

```
##      Up
## Down  0
## Up    1
```

```
glm.pred=rep("Down",1250)
glm.pred[glm.probs>.5]="Up"
table(glm.pred,Direction)
```

```
##         Direction
## glm.pred Down  Up
##     Down  145 141
##     Up    457 507
```

```
(507+145)/1250
```

```
## [1] 0.5216
```

```r
mean(glm.pred==Direction)
```

```
## [1] 0.5216
```

```r
train=(Year<2005)
Smarket.2005=Smarket[!train,]
dim(Smarket.2005)
```

```
## [1] 252   9
```

```r
Direction.2005=Direction[!train]
glm.fit=glm(
  Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
  data=Smarket,family=binomial,subset=train)
glm.probs=predict(glm.fit,Smarket.2005,type="response")
glm.pred=rep("Down",252)
glm.pred[glm.probs>.5]="Up"
table(glm.pred,Direction.2005)
```

```
##         Direction.2005
## glm.pred Down Up
##     Down   77 97
##     Up     34 44
```

```r
mean(glm.pred==Direction.2005)
```

```
## [1] 0.4801587
```

```r
mean(glm.pred!=Direction.2005)
```

```
## [1] 0.5198413
```

```r
glm.fit=glm(Direction~Lag1+Lag2,data=Smarket,family=binomial,subset=train)
glm.probs=predict(glm.fit,Smarket.2005,type="response")
glm.pred=rep("Down",252)
glm.pred[glm.probs>.5]="Up"
table(glm.pred,Direction.2005)
```

```
##         Direction.2005
## glm.pred Down  Up
##     Down   35  35
##     Up     76 106
```

```r
mean(glm.pred==Direction.2005)
```

```
## [1] 0.5595238
```

```r
106/(106+76)
```

```
## [1] 0.5824176
```

```r
predict(glm.fit,newdata=data.frame(Lag1=c(1.2,1.5),Lag2=c(1.1,-0.8)),type="response")
```

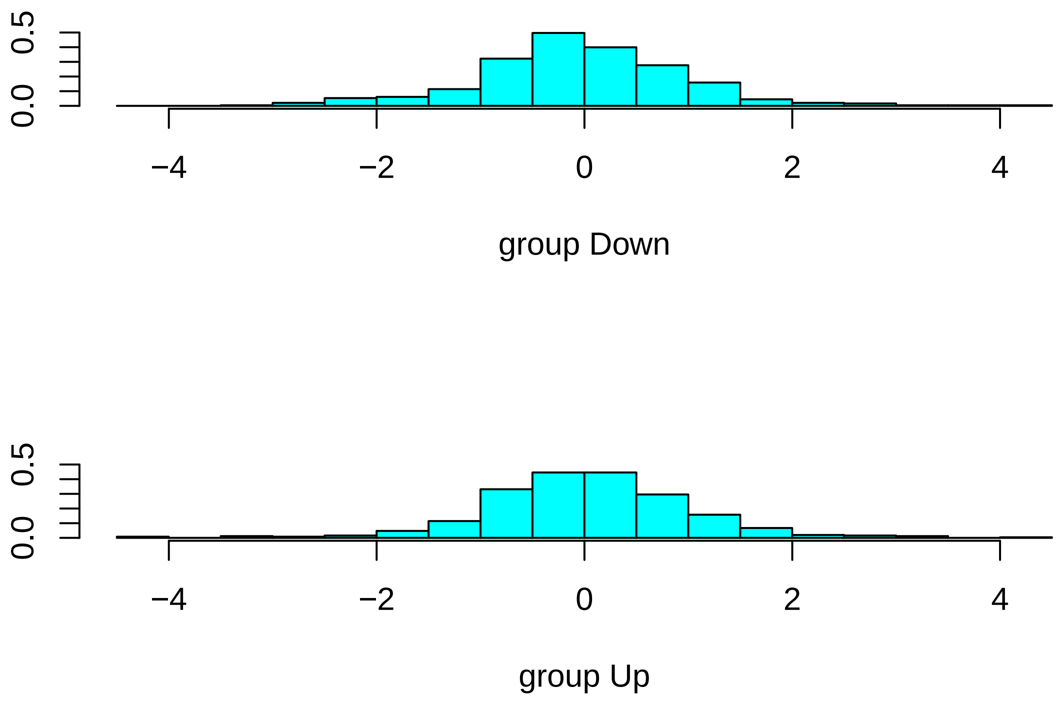
```
##         1         2
## 0.4791462 0.4960939
```

```r
# Linear Discriminant Analysis

library(MASS)
```

```r
lda.fit=lda(Direction~Lag1+Lag2,data=Smarket,subset=train)
lda.fit
```

```
## Call:
## lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
##
## Coefficients of linear discriminants:
##             LD1
## Lag1 -0.6420190
## Lag2 -0.5135293
```

```r
plot(lda.fit)
```



```r
lda.pred=predict(lda.fit, Smarket.2005)
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

```r
lda.class=lda.pred$class
table(lda.class,Direction.2005)
```

```
##          Direction.2005
## lda.class Down  Up
##      Down   35  35
##      Up     76 106
```

```r
mean(lda.class==Direction.2005)
```

```
## [1] 0.5595238
```

```r
sum(lda.pred$posterior[,1]>=.5)
```

```
## [1] 70
```

```r
sum(lda.pred$posterior[,1]<.5)
```

```
## [1] 182
```

```r
lda.pred$posterior[1:20,1]
```

```
##       999      1000      1001      1002      1003      1004      1005
## 0.4901792 0.4792185 0.4668185 0.4740011 0.4927877 0.4938562 0.4951016
##      1006      1007      1008      1009      1010      1011      1012
## 0.4872861 0.4907013 0.4844026 0.4906963 0.5119988 0.4895152 0.4706761
##      1013      1014      1015      1016      1017      1018
## 0.4744593 0.4799583 0.4935775 0.5030894 0.4978806 0.4886331
```

```r
lda.class[1:20]
```

```
##  [1] Up   Up   Up   Up   Up   Up   Up   Up   Up   Up   Up   Down Up   Up
## [15] Up   Up   Up   Down Up   Up
## Levels: Down Up
```

```r
sum(lda.pred$posterior[,1]>.9)
```

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

```r
# Quadratic Discriminant Analysis

qda.fit=qda(Direction~Lag1+Lag2,data=Smarket,subset=train)
qda.fit
```

```
## Call:
## qda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
```

```r
qda.class=predict(qda.fit,Smarket.2005)$class
table(qda.class,Direction.2005)
```

```
##          Direction.2005
## qda.class Down  Up
##      Down   30  20
##      Up     81 121
```

```
mean(qda.class==Direction.2005)
```

```
## [1] 0.5992063
```

```
# K-Nearest Neighbors

library(class)
train.X=cbind(Lag1,Lag2)[train,]
test.X=cbind(Lag1,Lag2)[!train,]
train.Direction=Direction[train]
set.seed(1)
knn.pred=knn(train.X,test.X,train.Direction,k=1)
table(knn.pred,Direction.2005)
```

```
##         Direction.2005
## knn.pred Down Up
##     Down   43 58
##     Up     68 83
```

```
(83+43)/252
```

```
## [1] 0.5
```

```
knn.pred=knn(train.X,test.X,train.Direction,k=3)
table(knn.pred,Direction.2005)
```

```
##         Direction.2005
## knn.pred Down Up
##     Down   48 54
##     Up     63 87
```

```
mean(knn.pred==Direction.2005)
```

```
## [1] 0.5357143
```

12.7 Application: Insurance Data

```
# An Application to Caravan Insurance Data

dim(Caravan)
```

```
## [1] 5822   86
```

```
attach(Caravan)
summary(Purchase)
```

```
##   No  Yes
## 5474  348
```

```
348/5822
```

```
## [1] 0.05977327
```

```
standardized.X=scale(Caravan[,-86])
var(Caravan[,1])
```

```
## [1] 165.0378
```

```
var(Caravan[,2])
```

```
## [1] 0.1647078
```

```
var(standardized.X[,1])
```

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

```
var(standardized.X[,2])
```

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

```
test=1:1000
train.X=standardized.X[-test,]
test.X=standardized.X[test,]
train.Y=Purchase[-test]
test.Y=Purchase[test]
set.seed(1)
knn.pred=knn(train.X,test.X,train.Y,k=1)
mean(test.Y!=knn.pred)
```

```
## [1] 0.118
```

```
mean(test.Y!="No")
```

```
## [1] 0.059
```

```
table(knn.pred,test.Y)
```

```
##         test.Y
## knn.pred  No Yes
##      No  873  50
##      Yes  68   9
```

```
9/(68+9)
```

```
## [1] 0.1168831
```

```
knn.pred=knn(train.X,test.X,train.Y,k=3)
table(knn.pred,test.Y)
```

```
##         test.Y
## knn.pred  No Yes
##      No  920  54
##      Yes  21   5
```

```
5/26
```

```
## [1] 0.1923077
```

```
knn.pred=knn(train.X,test.X,train.Y,k=5)
table(knn.pred,test.Y)
```

```
##         test.Y
## knn.pred  No Yes
##      No  930  55
##      Yes  11   4
```

```
4/15
```

```
## [1] 0.2666667
```

```r
glm.fit=glm(Purchase~.,data=Caravan,family=binomial,subset=-test)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
glm.probs=predict(glm.fit,Caravan[test,],type="response")
glm.pred=rep("No",1000)
glm.pred[glm.probs>.5]="Yes"
table(glm.pred,test.Y)
```

```
##         test.Y
## glm.pred  No Yes
##      No  934  59
##      Yes   7   0
```

```r
glm.pred=rep("No",1000)
glm.pred[glm.probs>.25]="Yes"
table(glm.pred,test.Y)
```

```
##         test.Y
## glm.pred  No Yes
##      No  919  48
##      Yes  22  11
```

```r
11/(22+11)
```

```
## [1] 0.3333333
```

13 Exercise 2

13.1 Boosting

13.1.1 Intuition

The key point, almost always missed in technical discussions, is that boosting is really about bias reduction. Take the linear model, our example in this posting. A linear model is rarely if ever exactly correct. Thus use of a linear model will result in bias; in some regions of the predictor vector X, the model will overestimate the true regression function, while in others it will underestimate - no matter how large our sample is. It thus may be profitable to try to reduce bias in regions in which our unweighted predictions are very bad, at the hopefully small sacrifice of some prediction accuracy in places where our unweighted analysis is doing well. (In the classification setting, a small loss in accuracy in estimating the conditional probability function won't hurt our predictions at all, since our predictions won't change.) The reweighting (or other iterative) process is aimed at achieving a positive tradeoff of that nature.

13.1.2 Model

In general context, consider a model like $\mathbb{E}[Y|\mathbf{X}=x]=H(x)$, and we write it as $\mathbb{E}[Y|\mathbf{X}=x]=\sum_{j=1}^{M}h_j(x)$, or $\mathbb{E}[Y|\mathbf{X}=x]=\sum_{j=1}^{M}v_ih_j(x)$ where v_i's will be some shrinkage parameters). To get all the components, we will use iterative procedure. Define the partial sum

$$H_j(x)=\sum_{k=1}^{j}h_k(x)$$

Since we consider some regression function here, use the l_2 loss function, to get the $h_j(\cdot)$ function, we solve

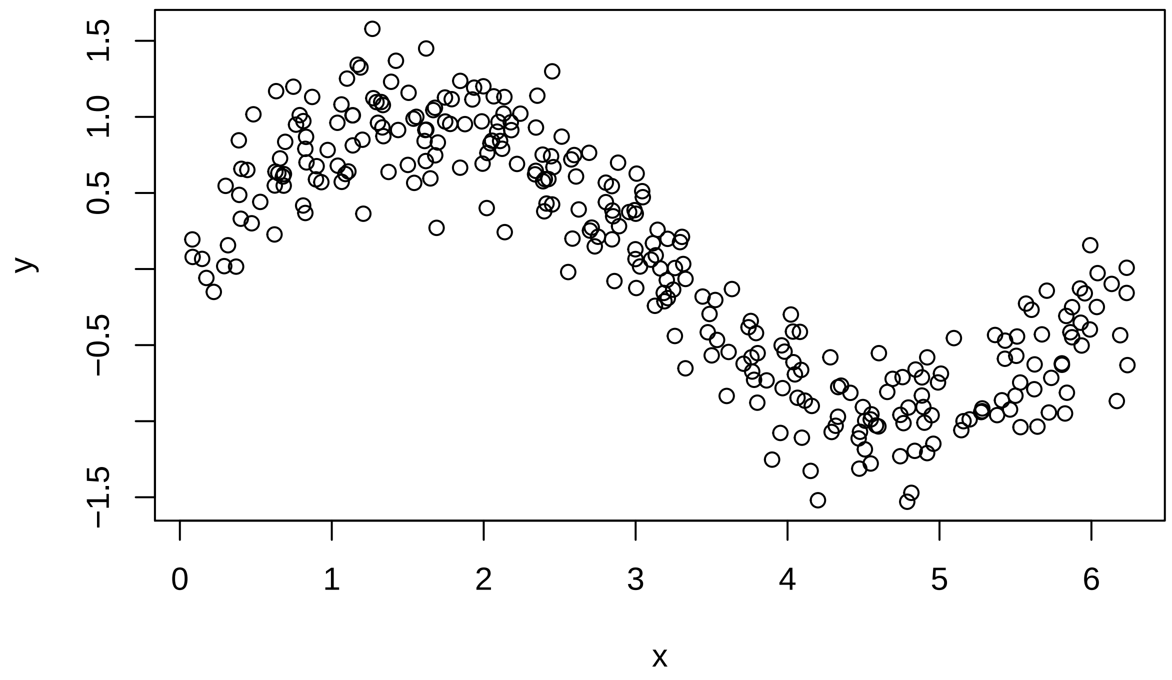
$$\min_{h(\cdot)}\left\{\sum_{i=1}^{n}[y_i-H_{j-1}(x_i)-h(x_i)]^2\right\}$$

and can imagine that the loss function can be changed (for classification instance).

The iterative algorithm is (1) start with some regression model $y_1=h_1(x)$, (2) compute the residuals, including some shrinkage parameter, $\epsilon_i=y-v_1h_1(x)$, (3) at step j, consider regression $\epsilon_j=h_j(x)$, (4) update the residuals $\epsilon_{j+1}=\epsilon_j-v_jh_j(x)$ and to loop. Then set

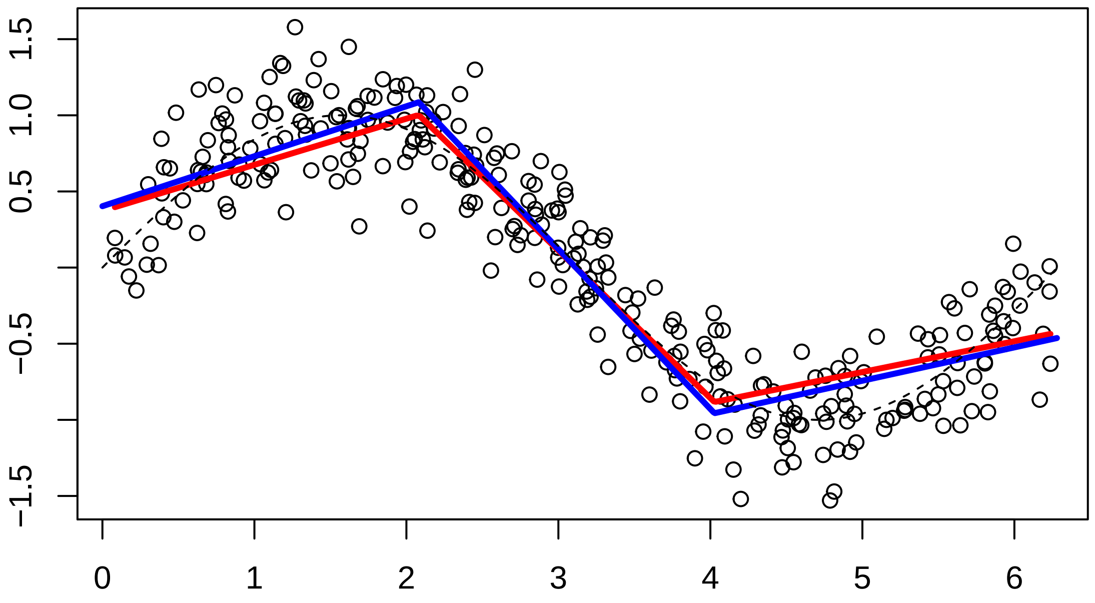
$$\hat{y}=\sum_{j=1}^{M}v_j\epsilon_j=\sum_{j=1}^{M}v_jh_j(x)$$

```
# Create sample data:
n=300
set.seed(1)
u=sort(runif(n)*2*pi)
y=sin(u)+rnorm(n)/4
df=data.frame(x=u,y=y)

# Visualize:
plot(df)
```



```
# Visualize:
# Red line is the initial guess
# we have, without boosting,
# using a simple call of the
# regression function. The blue
# one is the one obtained using
# boosting. The dotted line is the true model.
v=.05
library(splines)
fit=lm(y~bs(x,degree=1,df=3),data=df)
yp=predict(fit,newdata=df)
df$yr=df$y - v*yp
YP=v*yp

for(t in 1:100){
  fit=lm(yr~bs(x,degree=1,df=3),data=df)
  yp=predict(fit,newdata=df)
  df$yr=df$yr - v*yp
  YP=cbind(YP,v*yp)
}

nd=data.frame(x=seq(0,2*pi,by=.01))
viz=function(M){
if(M==1)  y=YP[,1]
if(M>1)   y=apply(YP[,1:M],1,sum)
  plot(df$x,df$y,ylab="",xlab="")
  lines(df$x,y,type="l",col="red",lwd=3)
  fit=lm(y~bs(x,degree=1,df=3),data=df)
  yp=predict(fit,newdata=nd)
  lines(nd$x,yp,type="l",col="blue",lwd=3)
  lines(nd$x,sin(nd$x),lty=2)}
viz(50)
```

```
## Warning in bs(x, degree = 1L, knots = structure(c(2.08092116216283,
## 4.02645437093874: some 'x' values beyond boundary knots may cause ill-
## conditioned bases
```



13.2 Dimension Reduction Techniques

```
# Use Swiss dataset for linear model:
head(swiss)
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
##              Infant.Mortality
## Courtelary               22.2
## Delemont                    22.2
## Franches-Mnt                20.2
## Moutier                     20.3
## Neuveville                  20.6
## Porrentruy                  26.6
```

```r
head(longley) # Use this data set as example!
```

```
##      GNP.deflator     GNP Unemployed Armed.Forces Population Year Employed
## 1947         83.0 234.289      235.6        159.0    107.608 1947   60.323
## 1948         88.5 259.426      232.5        145.6    108.632 1948   61.122
## 1949         88.2 258.054      368.2        161.6    109.773 1949   60.171
## 1950         89.5 284.599      335.1        165.0    110.929 1950   61.187
## 1951         96.2 328.975      209.9        309.9    112.075 1951   63.221
## 1952         98.1 346.999      193.2        359.4    113.270 1952   63.639
```

13.2.1 PCR

```r
require(pls)
```

```
## Loading required package: pls
```

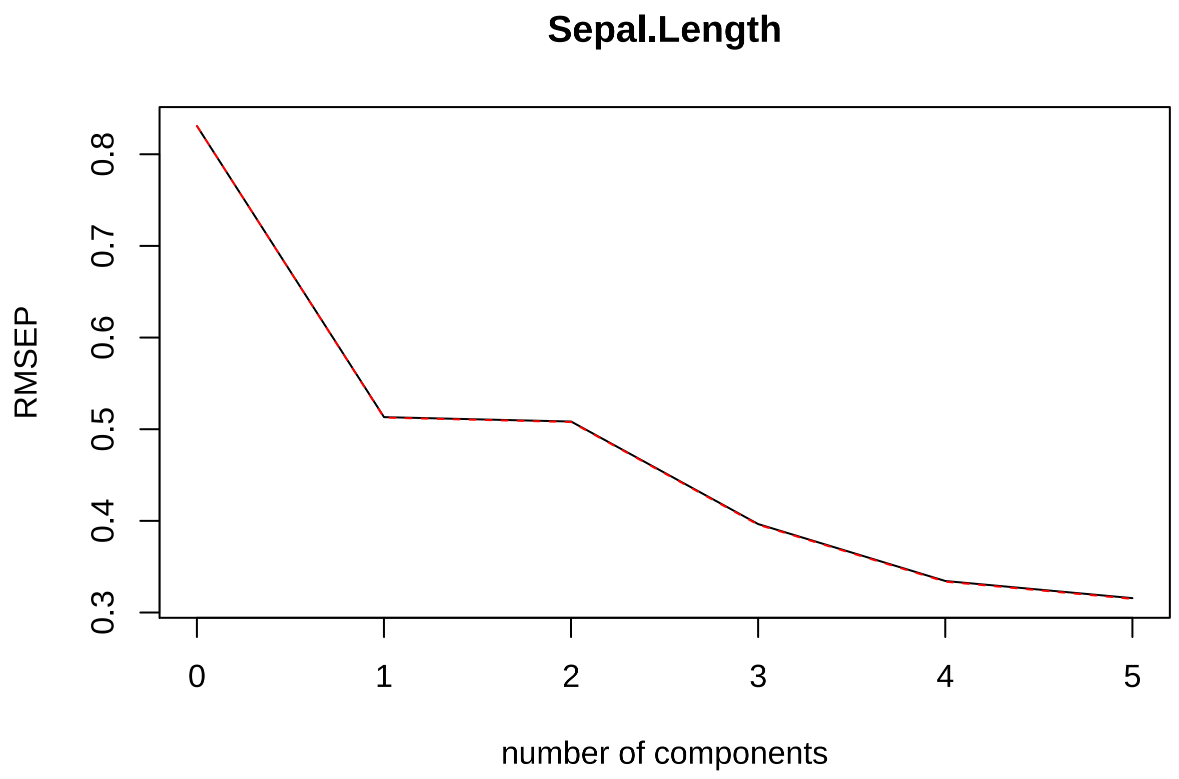
```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

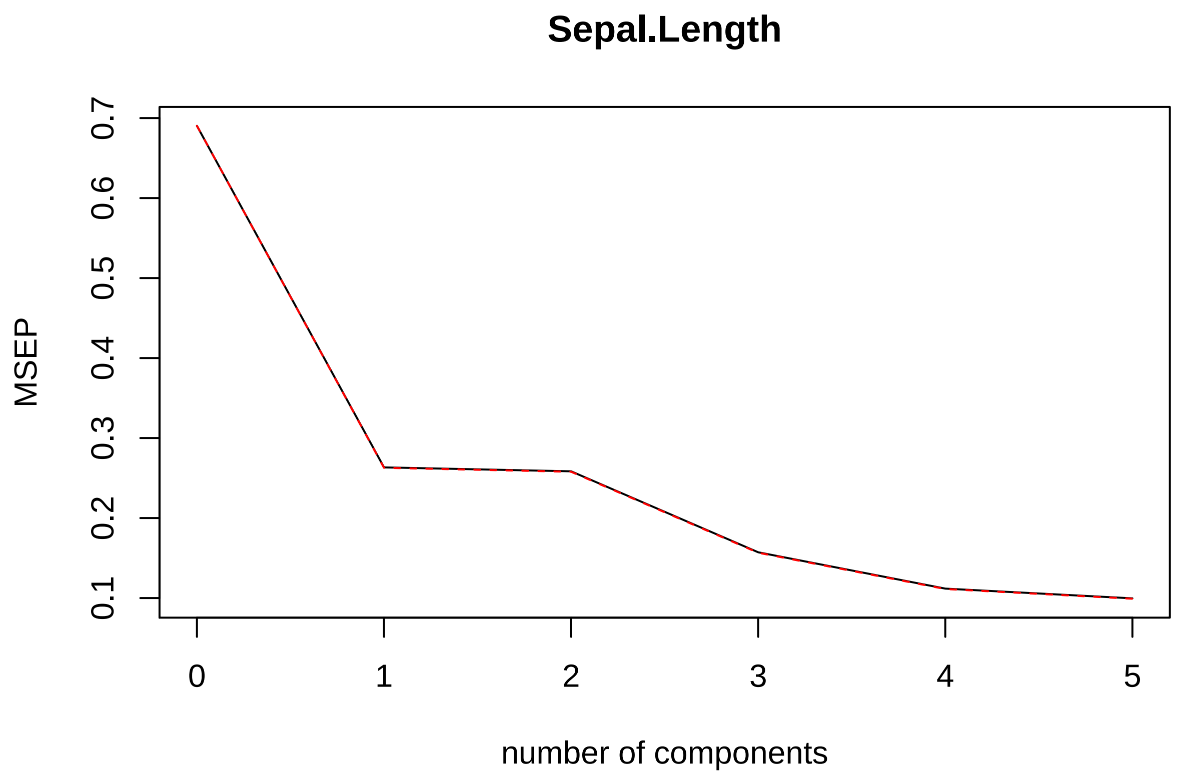
```r
pcr_model <- pcr(Sepal.Length~., data = iris, scale = TRUE, validation = "CV")
# Comment:
# By setting the parameter scale equal
# to TRUE the data is standardized before
# running the pcr algorithm on it. You can
# also perform validation by setting the
# argument validation. In this case I
# chose to perform 10 fold cross-validation
# and therefore set the validation argument
# to "CV", however there other validation
# methods available just type ?pcr in the
# R command window to gather some more
# information on the parameters of the pcr function.

# Suumary:
summary(pcr_model)
```

```
## Data:    X dimension: 150 5
##  Y dimension: 150 1
## Fit method: svdpc
## Number of components considered: 5
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps
## CV          0.8308   0.5132   0.5084   0.3965   0.3344   0.3157
## adjCV       0.8308   0.5126   0.5078   0.3958   0.3336   0.3149
##
## TRAINING: % variance explained
##               1 comps  2 comps  3 comps  4 comps  5 comps
## X               56.20    88.62    99.07    99.73   100.00
## Sepal.Length    62.71    63.58    78.44    84.95    86.73
```

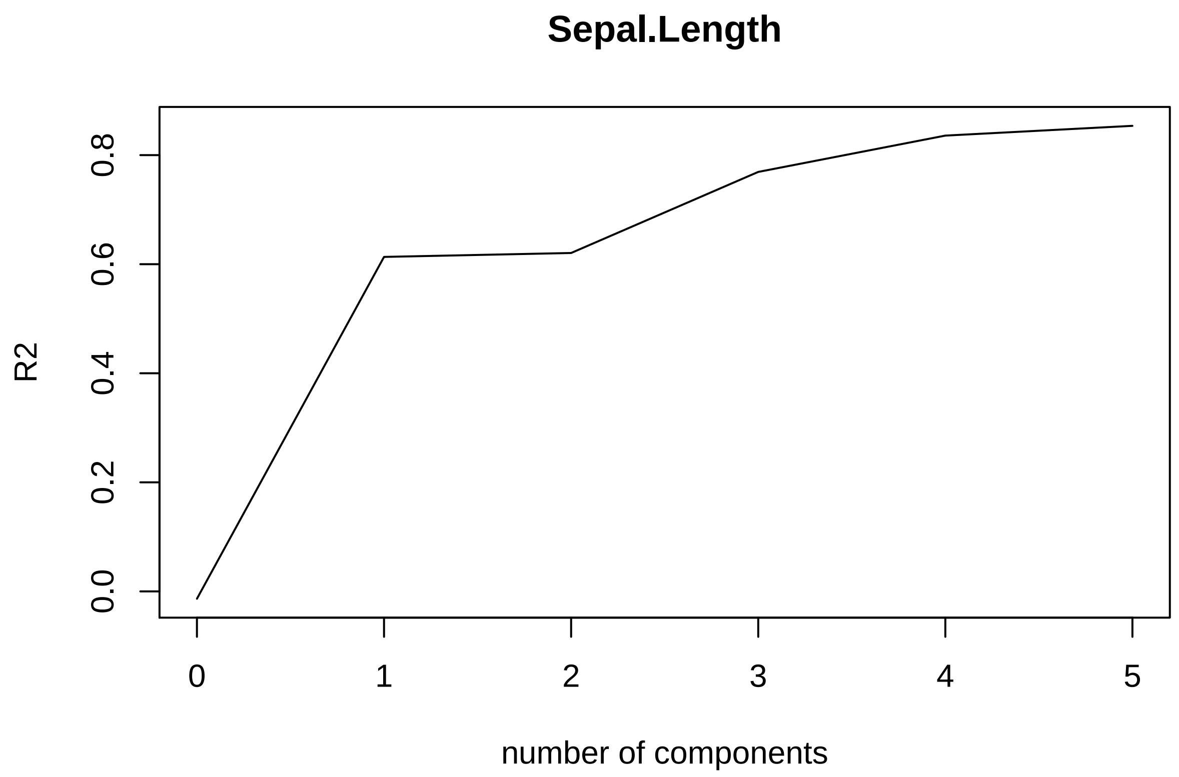
```
# Plot the root mean squared error
validationplot(pcr_model)
```



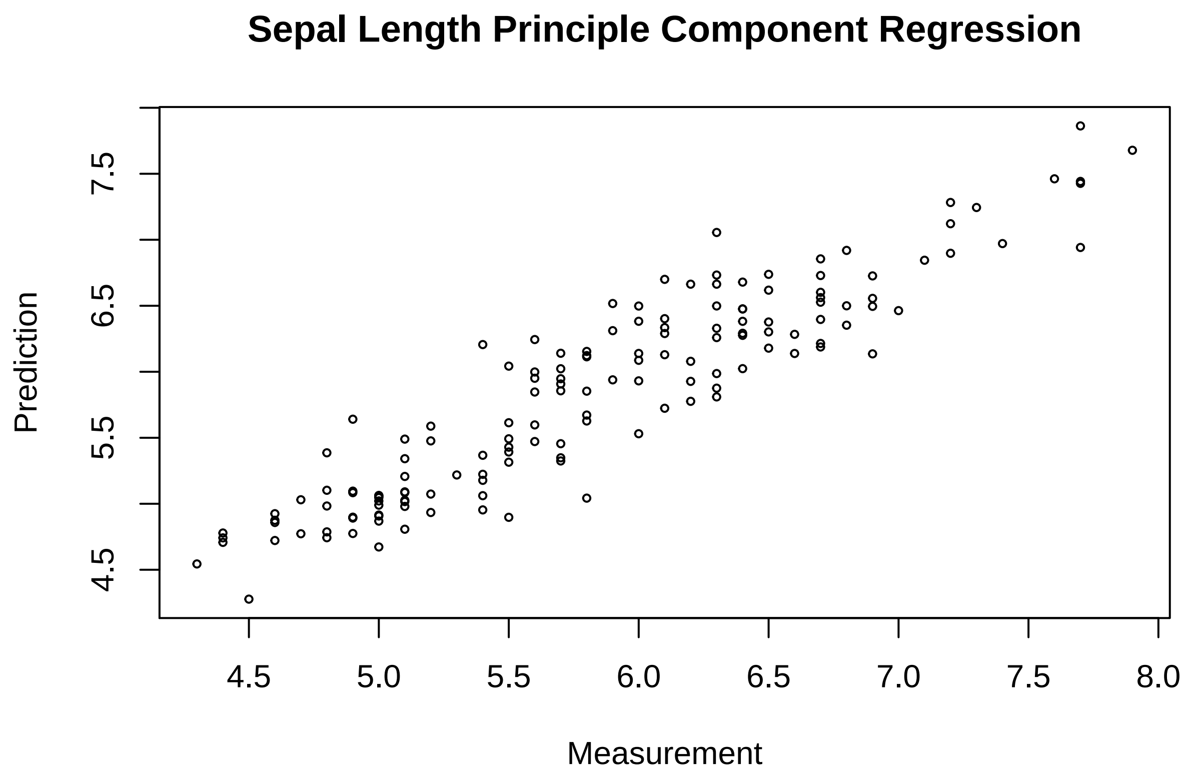
```
# Plot the cross validation MSE
validationplot(pcr_model, val.type="MSEP")
```



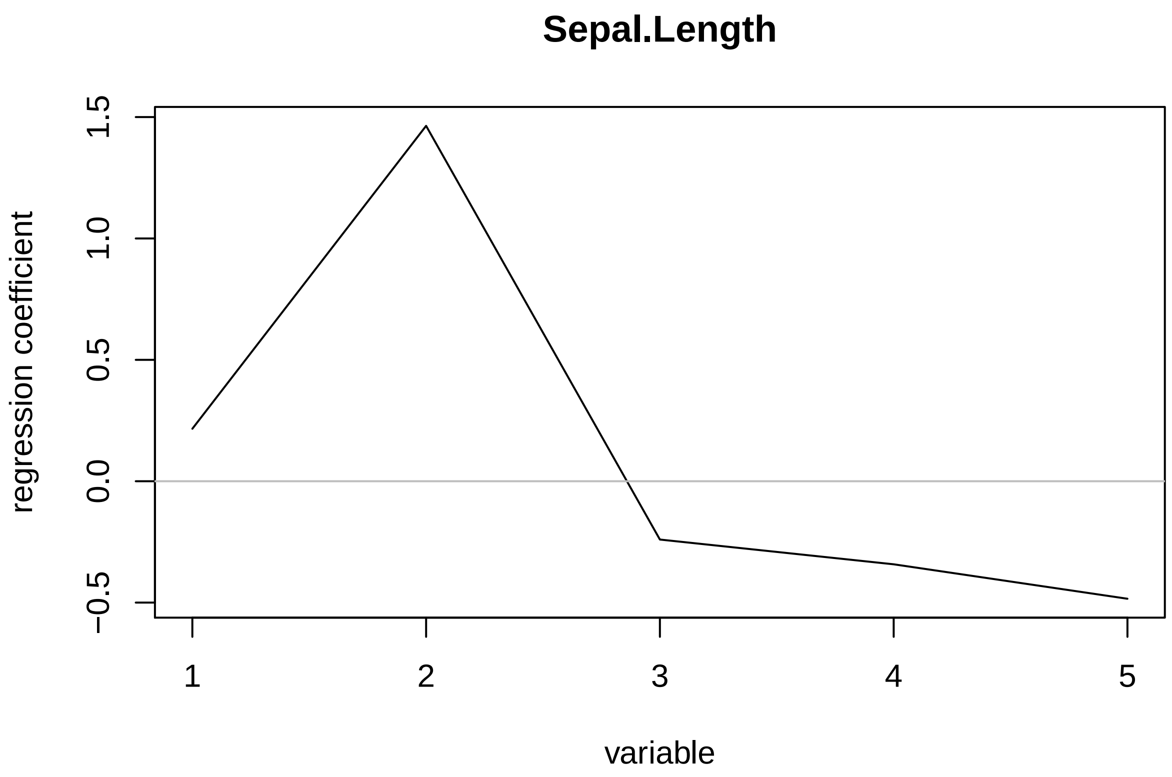
```
# Plot the R2
validationplot(pcr_model, val.type = "R2")
```



```
# Plot Prediction vs. Estimate
predplot(
  pcr_model,
  xlab="Measurement",
  ylab="Prediction",
  main="Sepal Length Principle Component Regression",
  cex=0.5)
```



```
# Plot Coefficients:
coefplot(pcr_model)
```



```
# Use PCR on a traning-test set
# and evaluate its performance
# using, for example, using only 3 components
# Train-test split
train <- iris[1:120,]
y_test <- iris[120:150, 1]
test <- iris[120:150, 2:5]

pcr_model <- pcr(Sepal.Length~., data = train,scale =TRUE, validation = "CV")

pcr_pred <- predict(pcr_model, test, ncomp = 3)
mean((pcr_pred - y_test)^2)
```

```
## [1] 0.213731
```

13.2.2 Step-wise Regression

```
fit <- lm(
    data=swiss,
    formula=swiss$Fertility~.
  )
step <- step(
  fit,
  direction="backward"
  # trace=0
)
```

```
## Start:  AIC=190.69
## swiss$Fertility ~ Agriculture + Examination + Education + Catholic +
##     Infant.Mortality
##
##                    Df Sum of Sq    RSS    AIC
## - Examination       1     53.03 2158.1 189.86
## <none>                          2105.0 190.69
## - Agriculture       1    307.72 2412.8 195.10
## - Infant.Mortality  1    408.75 2513.8 197.03
## - Catholic          1    447.71 2552.8 197.75
## - Education         1   1162.56 3267.6 209.36
##
## Step:  AIC=189.86
## swiss$Fertility ~ Agriculture + Education + Catholic + Infant.Mortality
##
##                    Df Sum of Sq    RSS    AIC
## <none>                          2158.1 189.86
## - Agriculture       1    264.18 2422.2 193.29
## - Infant.Mortality  1    409.81 2567.9 196.03
## - Catholic          1    956.57 3114.6 205.10
## - Education         1   2249.97 4408.0 221.43
```

```
step$anova
```

```
##            Step Df Deviance Resid. Df Resid. Dev      AIC
## 1               NA       NA        41   2105.043 190.6913
## 2 - Examination  1 53.02656        42   2158.069 189.8606
```

```
# Compare results
```

13.2.3 Ridge vs. Lasso

```
# Data:
swiss <- datasets::swiss # head(swiss)
x <- model.matrix(Fertility~., swiss)[,-1]
y <- swiss$Fertility
lambda <- 10^seq(10, -2, length = 100)

# Create test and training sets
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-10
```

```
set.seed(489)
train = sample(1:nrow(x), nrow(x)/2)
test = (-train)
ytest = y[test]

# OLS
```

```
swisslm <- lm(Fertility~., data = swiss)
coef(swisslm)
```

```
##      (Intercept)      Agriculture      Examination        Education
##       66.9151817       -0.1721140       -0.2580082       -0.8709401
##         Catholic Infant.Mortality
##        0.1041153        1.0770481
```

```
# Ridge
ridge.mod <- glmnet(x, y, alpha = 0, lambda = lambda)
predict(ridge.mod, s = 0, exact = T, type = 'coefficients')[1:6,]
```

```
##      (Intercept)      Agriculture      Examination        Education
##       66.9365901       -0.1721983       -0.2590771       -0.8705300
##         Catholic Infant.Mortality
##        0.1040307        1.0770215
```

```
swisslm <- lm(Fertility~., data = swiss, subset = train)
ridge.mod <- glmnet(x[train,], y[train], alpha = 0, lambda = lambda)

# Find the best lambda from our list via cross-validation
cv.out <- cv.glmnet(x[train,], y[train], alpha = 0)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold
```

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

# Make predictions
ridge.pred <- predict(ridge.mod, s = bestlam, newx = x[test,])
s.pred <- predict(swisslm, newdata = swiss[test,])

# Check MSE
mean((s.pred-ytest)^2)
```

```
## [1] 106.0087
```

```
mean((ridge.pred-ytest)^2)
```

```
## [1] 93.02157
```

```
# Take a look at the coefficients
out = glmnet(x[train,],y[train],alpha = 0)
predict(ridge.mod, type = "coefficients", s = bestlam)[1:6,]
```

```
##      (Intercept)      Agriculture      Examination        Education
##      64.90631178      -0.16557837      -0.59425090      -0.35814759
##         Catholic Infant.Mortality
##       0.06545382       1.30375306
```

```
# Lasso
lasso.mod <- glmnet(x[train,], y[train], alpha = 1, lambda = lambda)
lasso.pred <- predict(lasso.mod, s = bestlam, newx = x[test,])
mean((lasso.pred-ytest)^2)
```

```
## [1] 124.1039
```

```
lasso.coef  <- predict(lasso.mod, type = 'coefficients', s = bestlam)[1:6,]
lasso.coef
```

```
##      (Intercept)      Agriculture      Examination        Education
##      54.72576032      -0.01493362      -0.40726342      -0.05839363
##         Catholic Infant.Mortality
##       0.03829186       1.19563533
```

```
require(glmnet)
# Data = considering that we have a data frame named dataF, with its first column being the class
dataF <- swiss; head(swiss)
```

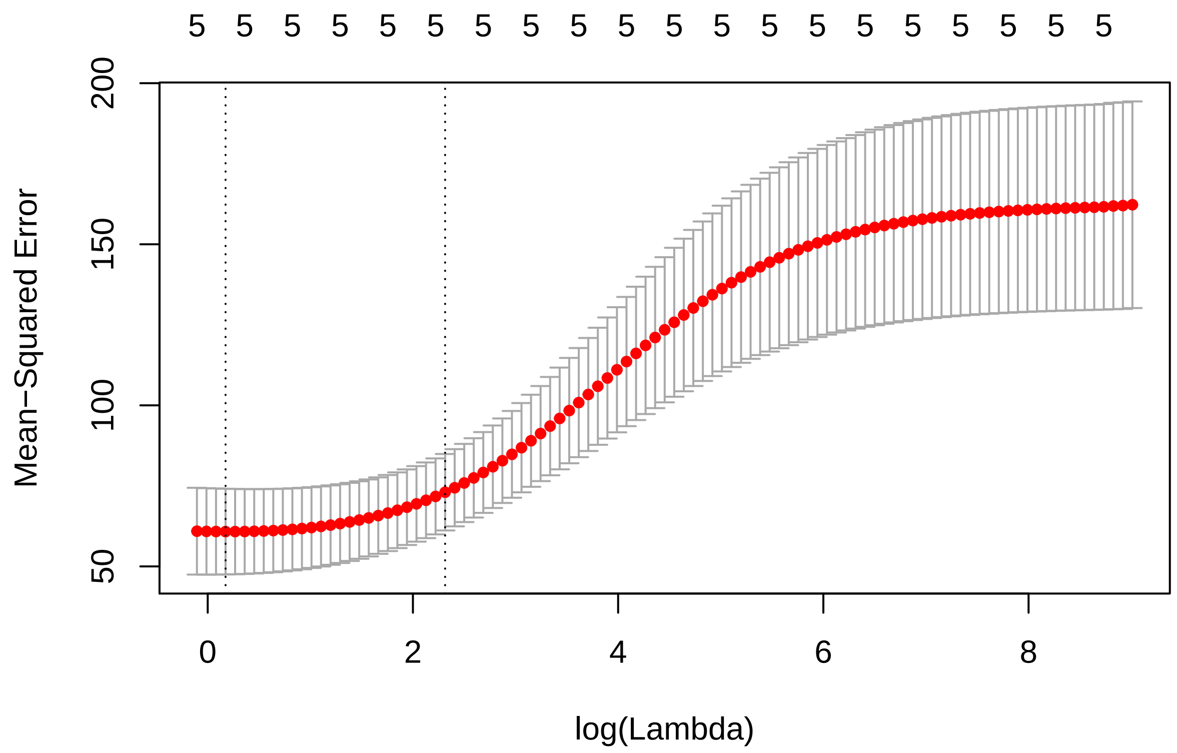
```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
##              Infant.Mortality
## Courtelary               22.2
## Delemont                 22.2
## Franches-Mnt             20.2
## Moutier                  20.3
## Neuveville               20.6
## Porrentruy               26.6
```

```
# Ridge
x <- as.matrix(dataF[,-1]) # Removes class
y <- as.double(as.matrix(dataF[, 1])) # Only class

# Fitting the model (Ridge: Alpha = 0)
set.seed(999)
cv.ridge <- cv.glmnet(
  x, y,
  family='gaussian', alpha=0,
  parallel=TRUE, standardize=TRUE,
  type.measure='auc')
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
## Warning in cv.elnet(list(structure(list(a0 = structure(c(70.852380952381, :
## Only 'mse', 'deviance' or 'mae' available for Gaussian models; 'mse' used
```

```
# Results
plot(cv.ridge)
```



```
cv.ridge$lambda.min
```

```
## [1] 1.190139
```

```
cv.ridge$lambda.1se
```

```
## [1] 10.11324
```

```
coef(cv.ridge, s=cv.ridge$lambda.min)
```

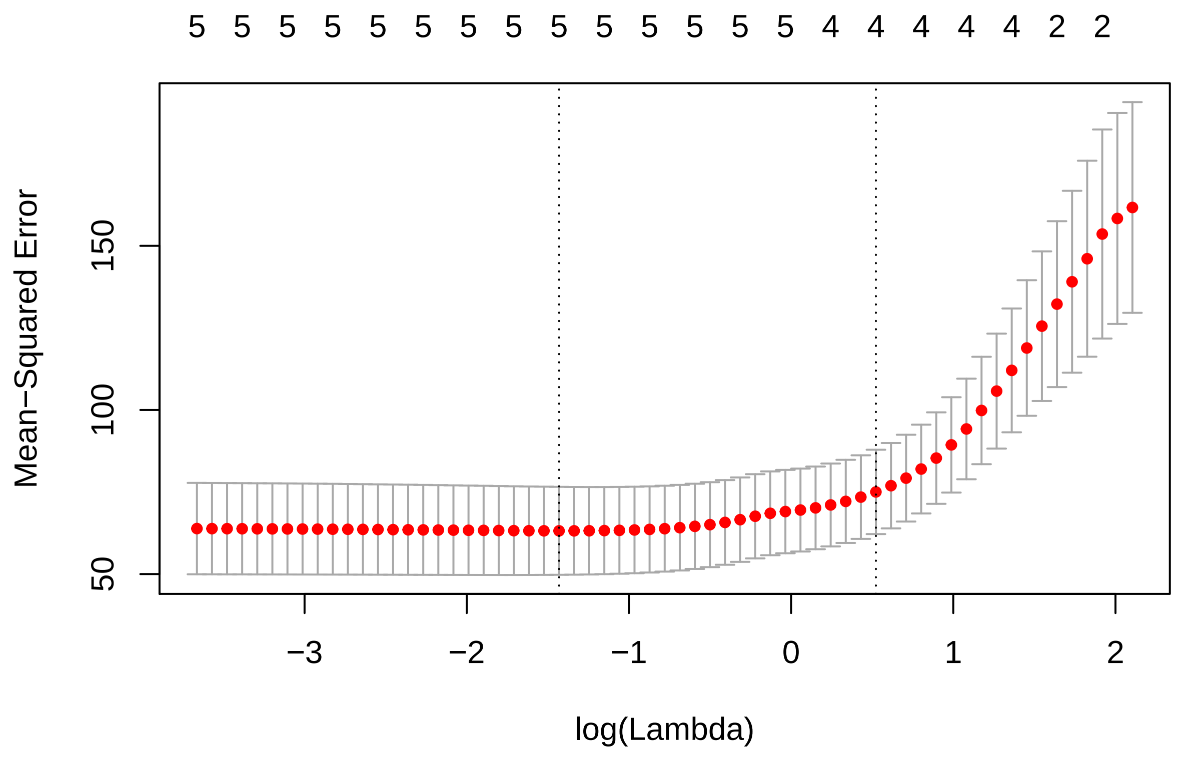
```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)      63.66015150
## Agriculture      -0.11232379
## Examination      -0.33164460
## Education        -0.68644253
## Catholic          0.08147413
## Infant.Mortality  1.09441301
```

```
# Here we use gaussian assuming linearity for the dataset we want to model.

# For the above code, we can also execute logistic regression (note the family='binomial'), in parallel

require(glmnet)
# Data = considering that we have a data frame named dataF, with its first column being the class
x <- as.matrix(dataF[,-1]) # Removes class
y <- as.double(as.matrix(dataF[, 1])) # Only class
```

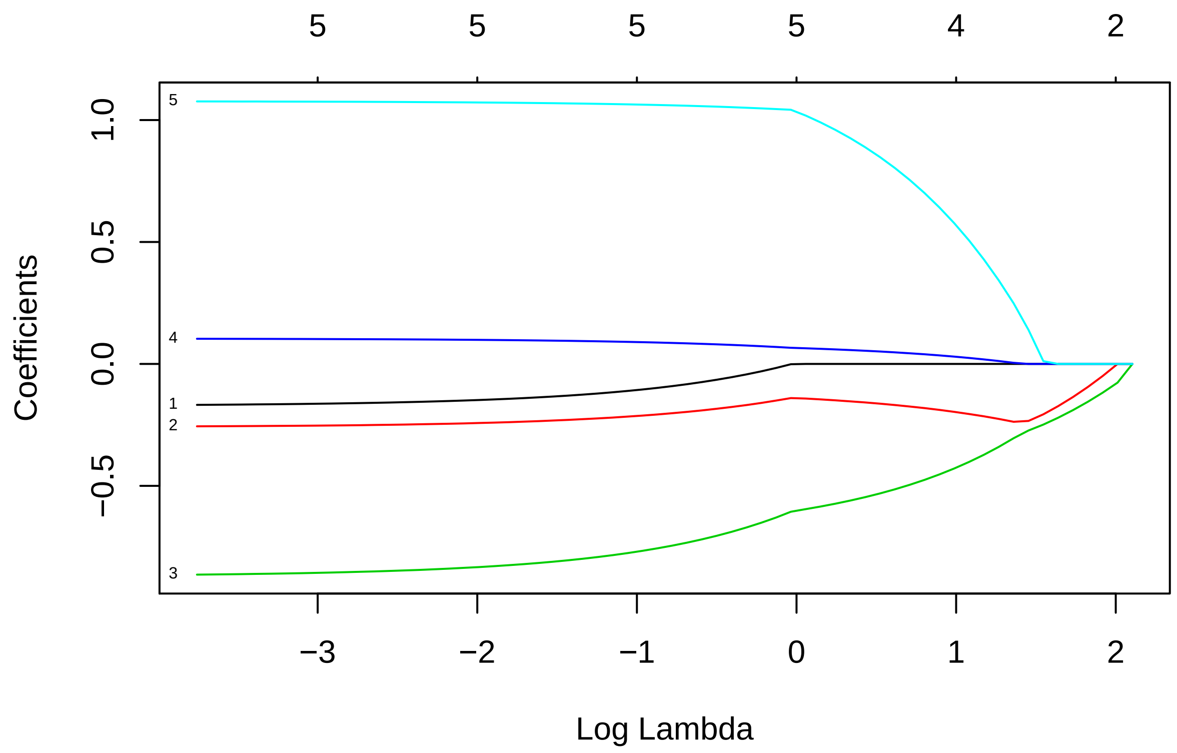
```r
# Lasso
# Fitting the model (Lasso: Alpha = 1)
set.seed(999)
cv.lasso <- cv.glmnet(
  x, y,
  family='gaussian',
  alpha=1, parallel=TRUE, standardize=TRUE,
  type.measure='auc')
```

```
## Warning in cv.elnet(list(structure(list(a0 = structure(c(70.852380952381, :
## Only 'mse', 'deviance' or 'mae' available for Gaussian models; 'mse' used
```

```r
# Results
plot(cv.lasso)
```



```r
plot(cv.lasso$glmnet.fit, xvar="lambda", label=TRUE)
```



```
cv.lasso$lambda.min
```

```
## [1] 0.2391266
```

```
cv.lasso$lambda.1se
```

```
## [1] 1.686991
```

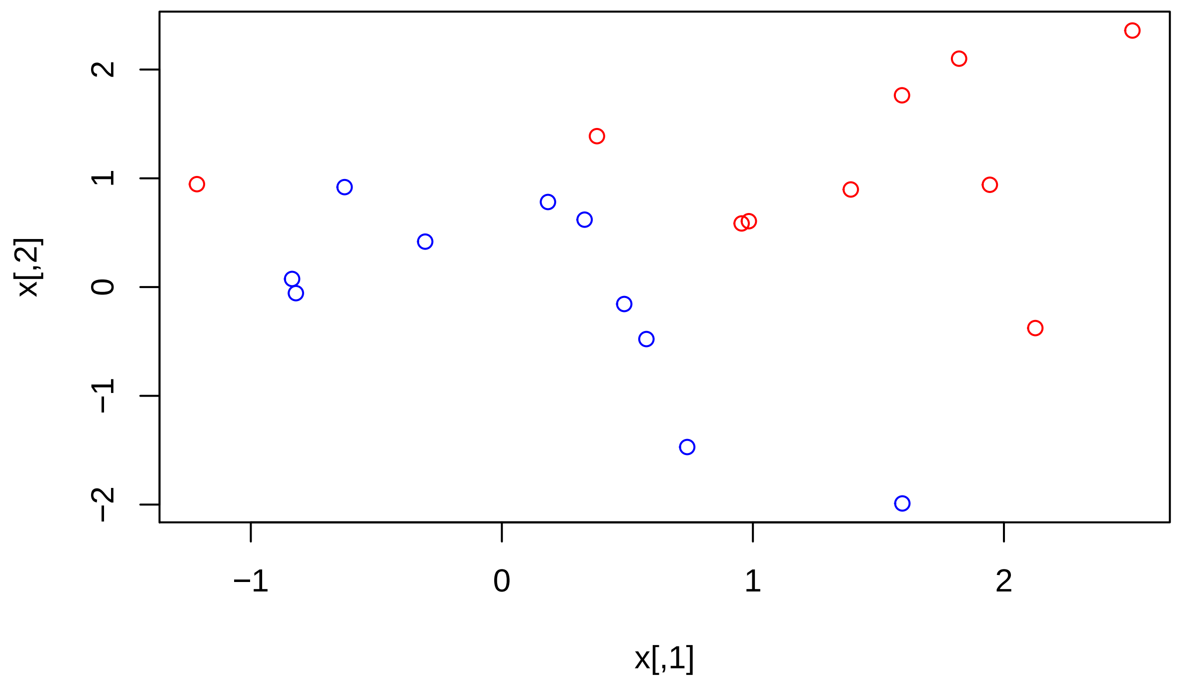
```
coef(cv.lasso, s=cv.lasso$lambda.min)
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                            1
## (Intercept)      64.14777592
## Agriculture      -0.12990878
## Examination      -0.22949180
## Education        -0.80516212
## Catholic          0.09466401
## Infant.Mortality  1.06831289
```

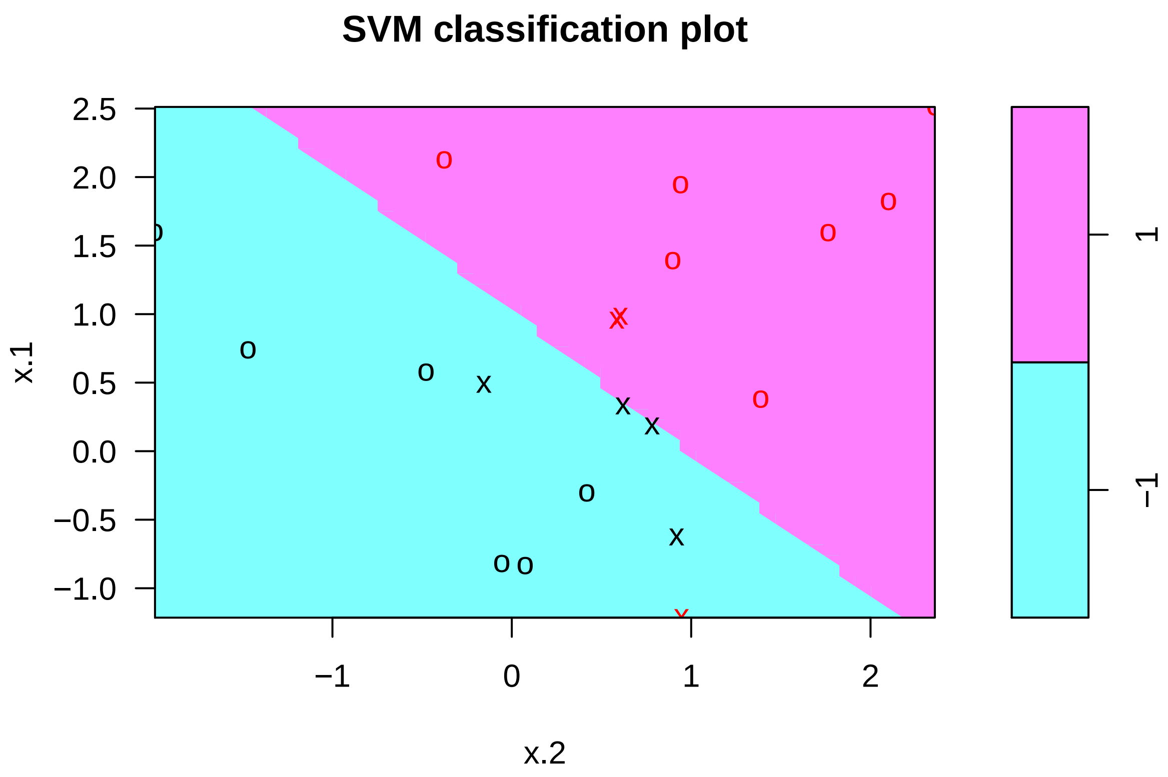
14 Exercise 3

14.1 Support Vector Classifier

```r
# Support Vector Classifier

set.seed(1)
x=matrix(rnorm(20*2), ncol=2)
y=c(rep(-1,10), rep(1,10))
x[y==1,]=x[y==1,] + 1
plot(x, col=(3-y))
```



```r
dat=data.frame(x=x, y=as.factor(y))
library(e1071)
svmfit=svm(y~., data=dat, kernel="linear", cost=10,scale=FALSE)
plot(svmfit, dat)
```



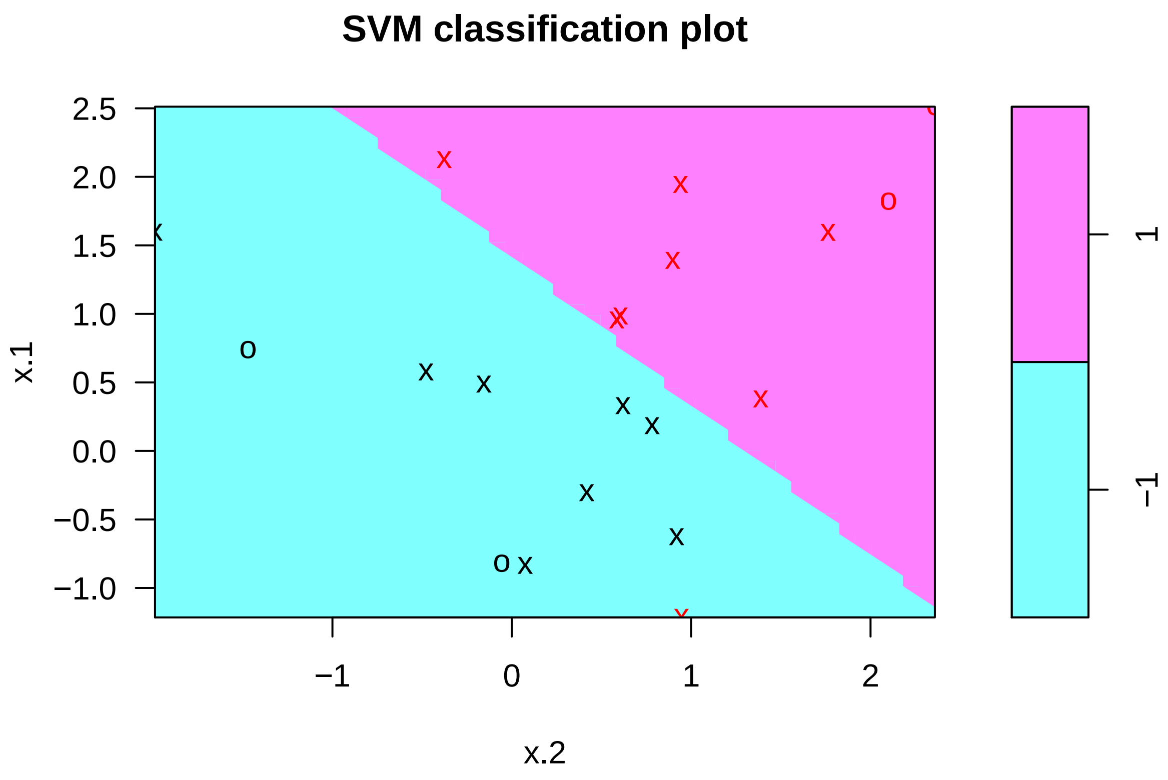
```
svmfit$index
```

```
## [1]  1  2  5  7 14 16 17
```

```
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
svmfit=svm(y~., data=dat, kernel="linear", cost=0.1,scale=FALSE)
plot(svmfit, dat)
```



```
svmfit$index
```

```
##  [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
```

```
set.seed(1)
tune.out=tune(svm,y~.,data=dat,kernel="linear",ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.70  0.4216370
## 2 1e-02  0.70  0.4216370
## 3 1e-01  0.10  0.2108185
## 4 1e+00  0.15  0.2415229
## 5 5e+00  0.15  0.2415229
## 6 1e+01  0.15  0.2415229
## 7 1e+02  0.15  0.2415229
```

```
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##       gamma:  0.5
##
## Number of Support Vectors:  16
##
##  ( 8 8 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

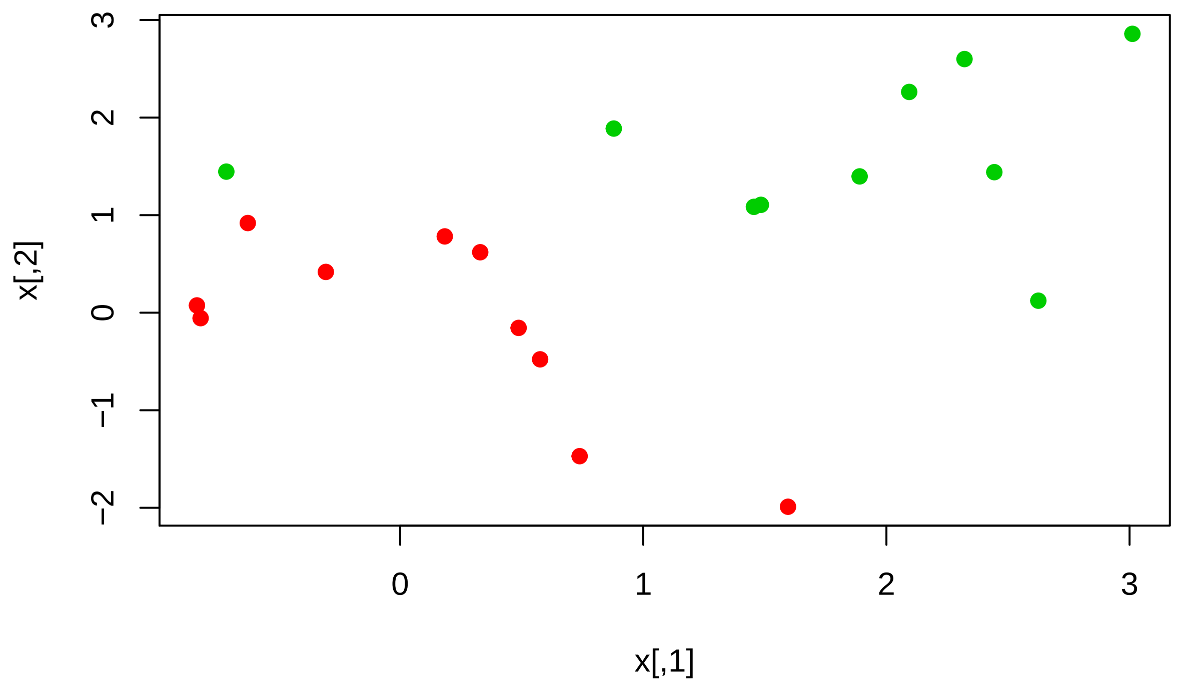
```
xtest=matrix(rnorm(20*2), ncol=2)
ytest=sample(c(-1,1), 20, rep=TRUE)
xtest[ytest==1,]=xtest[ytest==1,] + 1
testdat=data.frame(x=xtest, y=as.factor(ytest))
ypred=predict(bestmod,testdat)
table(predict=ypred, truth=testdat$y)
```

```
##        truth
## predict -1  1
##      -1 11  1
##      1   0  8
```

```
svmfit=svm(y~., data=dat, kernel="linear", cost=.01,scale=FALSE)
ypred=predict(svmfit,testdat)
table(predict=ypred, truth=testdat$y)
```

```
##        truth
## predict -1  1
##      -1 11  2
##      1   0  7
```

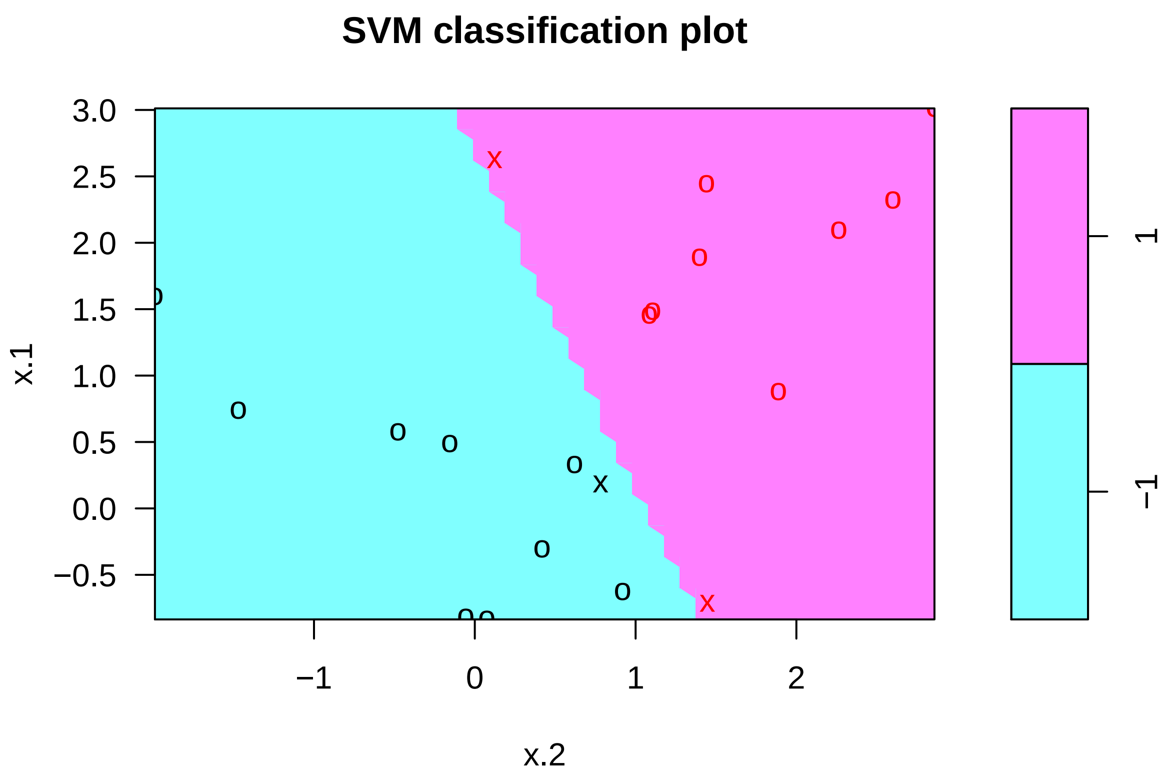
```
x[y==1,]=x[y==1,]+0.5
plot(x, col=(y+5)/2, pch=19)
```



```r
dat=data.frame(x=x,y=as.factor(y))
svmfit=svm(y~., data=dat, kernel="linear", cost=1e5)
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1e+05)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1e+05
##       gamma:  0.5
##
## Number of Support Vectors:  3
##
##  ( 1 2 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

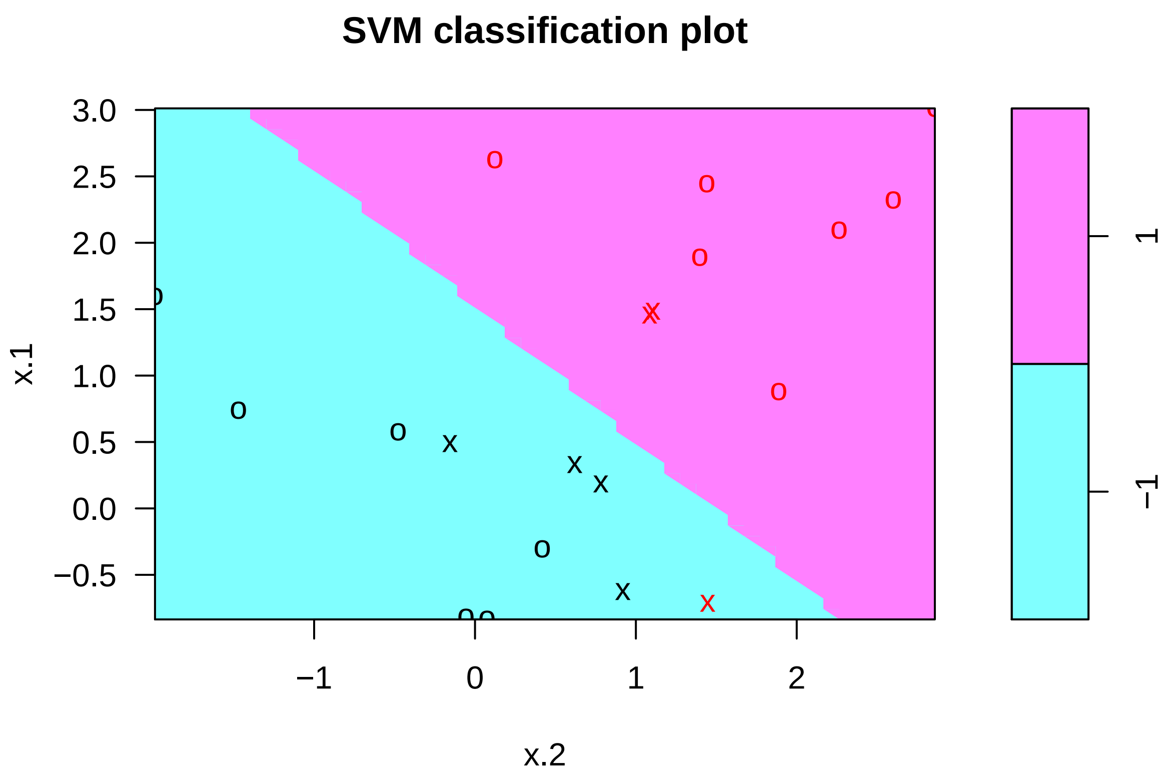
```r
plot(svmfit, dat)
```



```r
svmfit=svm(y~., data=dat, kernel="linear", cost=1)
summary(svmfit)
```

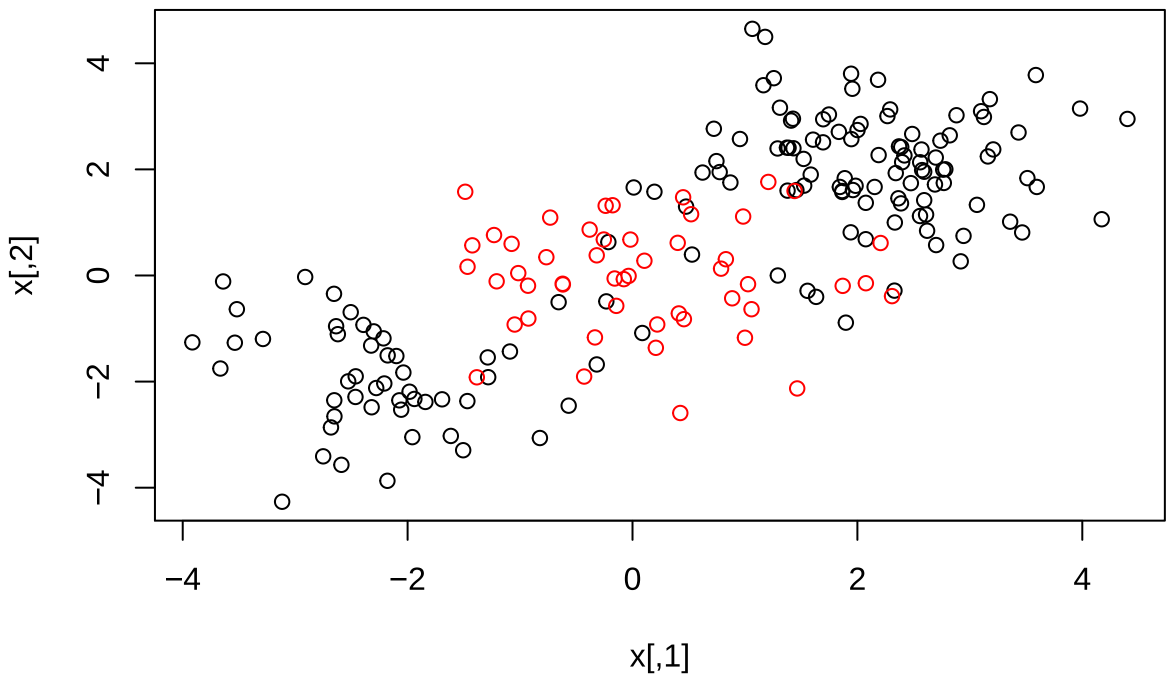
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.5
##
## Number of Support Vectors:  7
##
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```
plot(svmfit,dat)
```

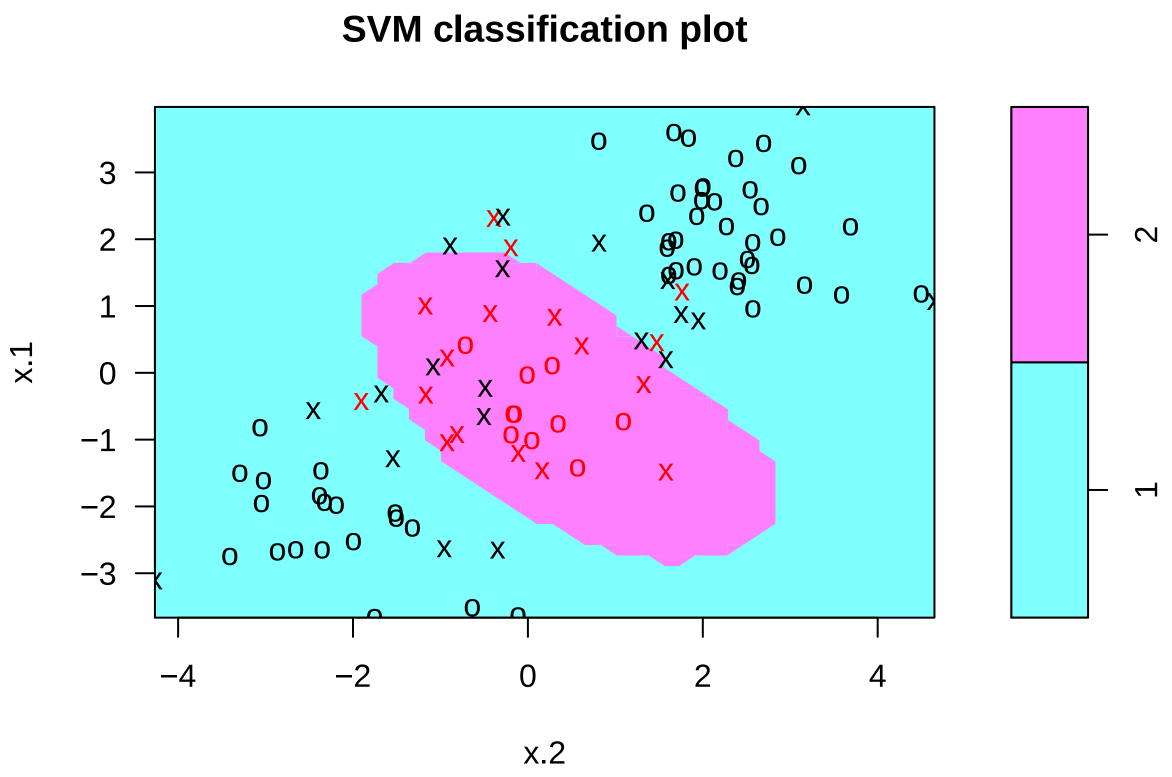


14.2 Support Vector Machine

```
# Support Vector Machine
set.seed(1)
x=matrix(rnorm(200*2), ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
plot(x, col=y)
```



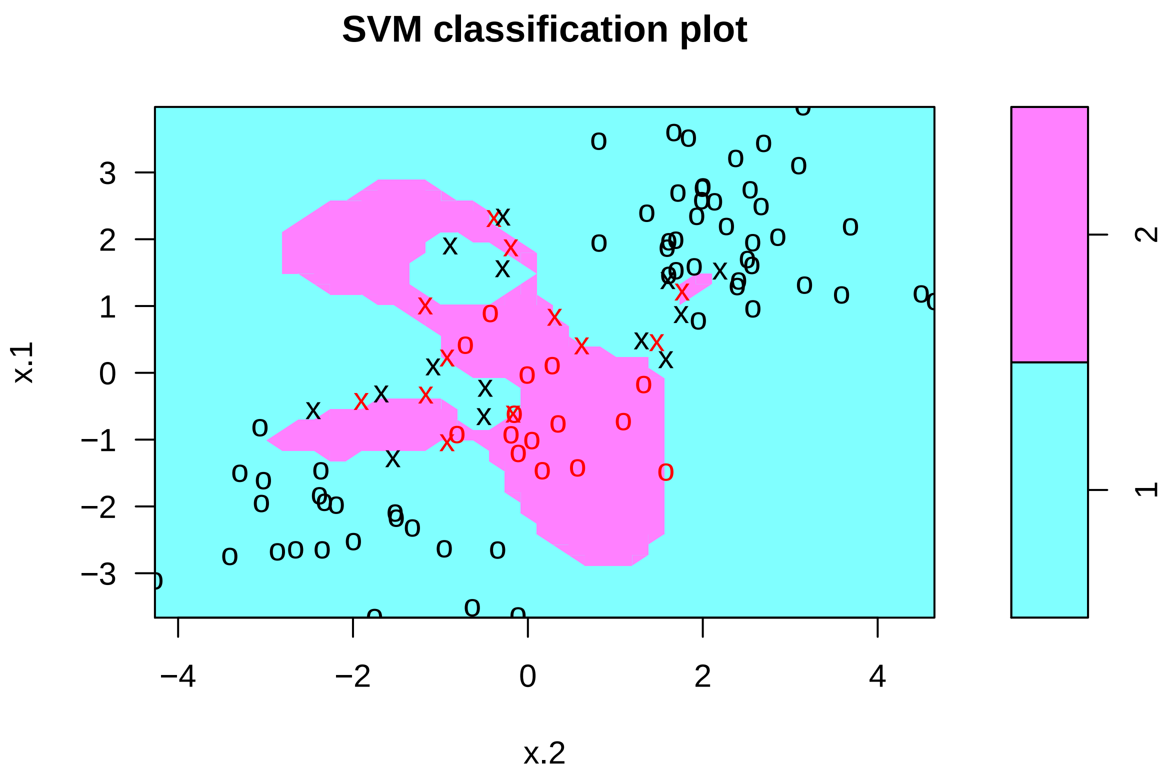
```
train=sample(200,100)
svmfit=svm(y~., data=dat[train,], kernel="radial",  gamma=1, cost=1)
plot(svmfit, dat[train,])
```



```r
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat[train, ], kernel = "radial",
##     gamma = 1, cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  37
##
##  ( 17 20 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

```r
svmfit=svm(y~., data=dat[train,], kernel="radial",gamma=1,cost=1e5)
plot(svmfit,dat[train,])
```



```r
set.seed(1)
tune.out=tune(svm, y~., data=dat[train,], kernel="radial", ranges=list(cost=c(0.1,1,10,100,1000),gamma=c
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     2
##
## - best performance: 0.12
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.27 0.11595018
## 2  1e+00   0.5  0.13 0.08232726
## 3  1e+01   0.5  0.15 0.07071068
## 4  1e+02   0.5  0.17 0.08232726
## 5  1e+03   0.5  0.21 0.09944289
## 6  1e-01   1.0  0.25 0.13540064
## 7  1e+00   1.0  0.13 0.08232726
## 8  1e+01   1.0  0.16 0.06992059
## 9  1e+02   1.0  0.20 0.09428090
## 10 1e+03   1.0  0.20 0.08164966
## 11 1e-01   2.0  0.25 0.12692955
## 12 1e+00   2.0  0.12 0.09189366
## 13 1e+01   2.0  0.17 0.09486833
## 14 1e+02   2.0  0.19 0.09944289
## 15 1e+03   2.0  0.20 0.09428090
## 16 1e-01   3.0  0.27 0.11595018
## 17 1e+00   3.0  0.13 0.09486833
## 18 1e+01   3.0  0.18 0.10327956
## 19 1e+02   3.0  0.21 0.08755950
## 20 1e+03   3.0  0.22 0.10327956
## 21 1e-01   4.0  0.27 0.11595018
## 22 1e+00   4.0  0.15 0.10801234
## 23 1e+01   4.0  0.18 0.11352924
## 24 1e+02   4.0  0.21 0.08755950
## 25 1e+03   4.0  0.24 0.10749677
```

```
table(true=dat[-train,"y"], pred=predict(tune.out$best.model,newx=dat[-train,]))
```

```
##     pred
## true  1  2
##    1 56 21
##    2 18  5
```

14.3 ROC Curve

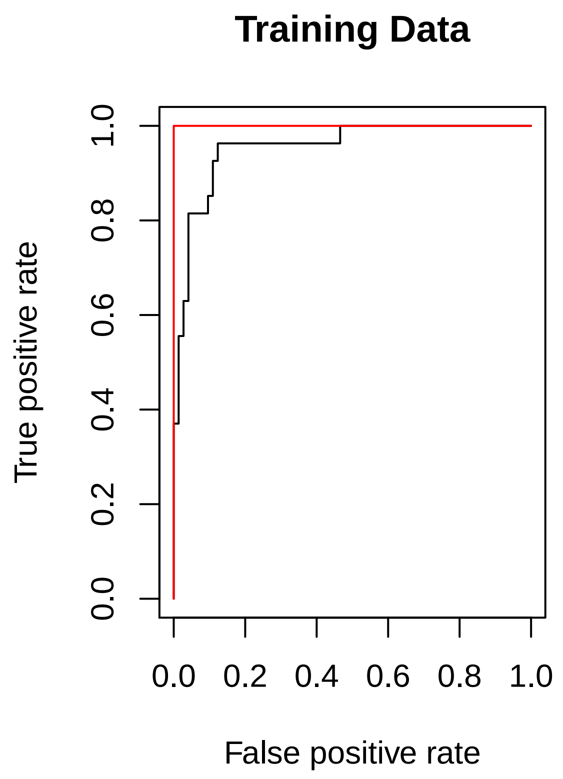
```
# ROC Curves

library(ROCR)
```

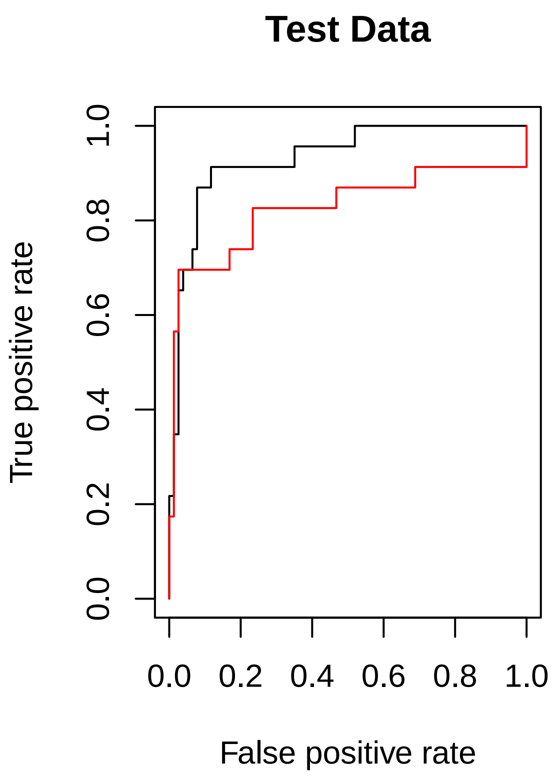
```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

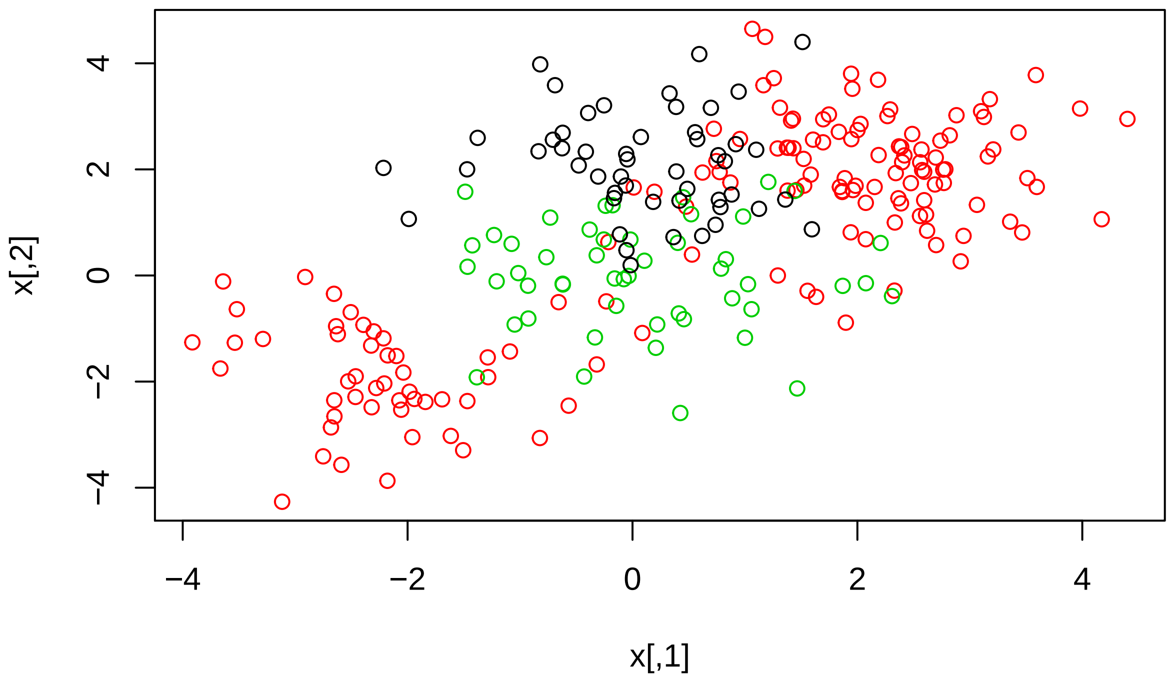
```
rocplot=function(pred, truth, ...){
   predob = prediction(pred, truth)
   perf = performance(predob, "tpr", "fpr")
   plot(perf,...)}
svmfit.opt=svm(y~., data=dat[train,], kernel="radial",gamma=2, cost=1,decision.values=T)
fitted=attributes(predict(svmfit.opt,dat[train,],decision.values=TRUE))$decision.values
par(mfrow=c(1,2))
rocplot(fitted,dat[train,"y"],main="Training Data")
svmfit.flex=svm(y~., data=dat[train,], kernel="radial",gamma=50, cost=1, decision.values=T)
fitted=attributes(predict(svmfit.flex,dat[train,],decision.values=T))$decision.values
rocplot(fitted,dat[train,"y"],add=T,col="red")
fitted=attributes(predict(svmfit.opt,dat[-train,],decision.values=T))$decision.values
rocplot(fitted,dat[-train,"y"],main="Test Data")
fitted=attributes(predict(svmfit.flex,dat[-train,],decision.values=T))$decision.values
rocplot(fitted,dat[-train,"y"],add=T,col="red")
```



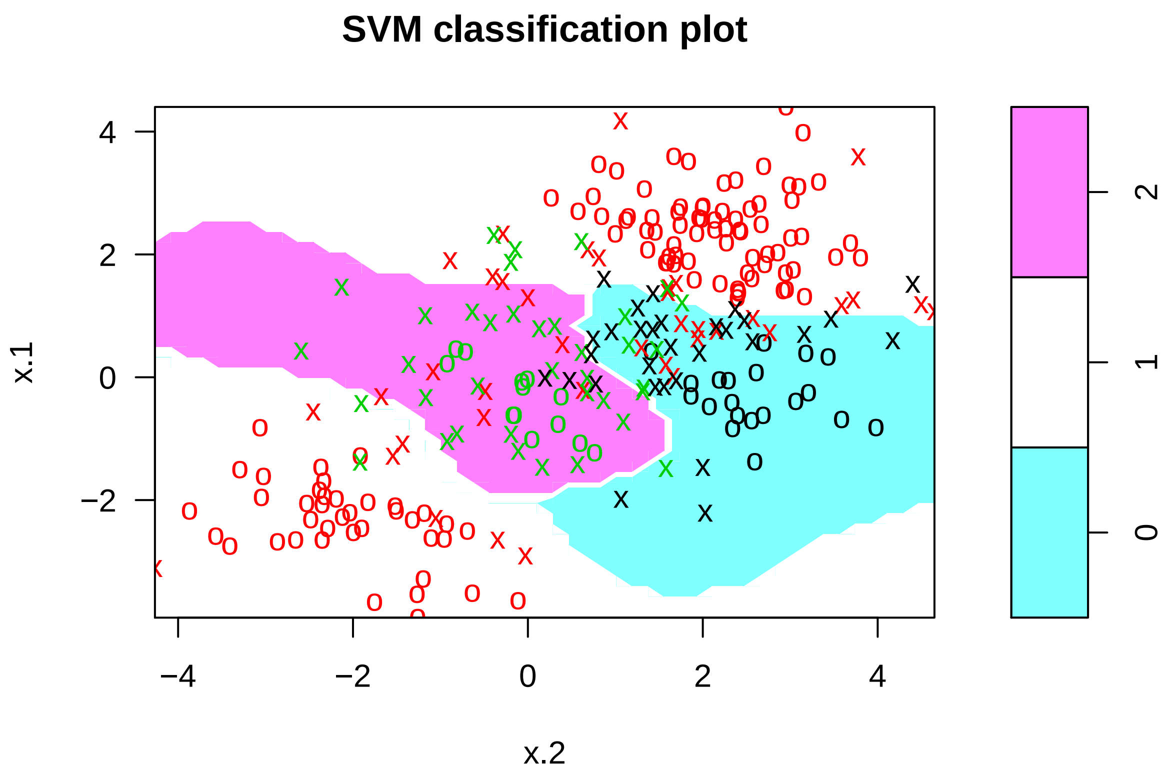


14.4 SVM with Multiple Classes

```
# SVM with Multiple Classes

set.seed(1)
x=rbind(x, matrix(rnorm(50*2), ncol=2))
y=c(y, rep(0,50))
x[y==0,2]=x[y==0,2]+2
dat=data.frame(x=x, y=as.factor(y))
par(mfrow=c(1,1))
plot(x,col=(y+1))
```



```
svmfit=svm(y~., data=dat, kernel="radial", cost=10, gamma=1)
plot(svmfit, dat)
```



14.5 Application to Gene Expression Data

```r
# Application to Gene Expression Data

library(ISLR)
names(Khan)
```

```
## [1] "xtrain" "xtest"  "ytrain" "ytest"
```

```r
dim(Khan$xtrain)
```

```
## [1]   63 2308
```

```r
dim(Khan$xtest)
```

```
## [1]   20 2308
```

```r
length(Khan$ytrain)
```

```
## [1] 63
```

```r
length(Khan$ytest)
```

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

```r
table(Khan$ytrain)
```

```
##
##  1  2  3  4
##  8 23 12 20
```

```
table(Khan$ytest)
```

```
##
## 1 2 3 4
## 3 6 6 5
```

```
dat=data.frame(x=Khan$xtrain, y=as.factor(Khan$ytrain))
out=svm(y~., data=dat, kernel="linear",cost=10)
summary(out)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.0004332756
##
## Number of Support Vectors:  58
##
##  ( 20 20 11 7 )
##
##
## Number of Classes:  4
##
## Levels:
##  1 2 3 4
```

```
table(out$fitted, dat$y)
```

```
##
##       1  2  3  4
##   1   8  0  0  0
##   2   0 23  0  0
##   3   0  0 12  0
##   4   0  0  0 20
```

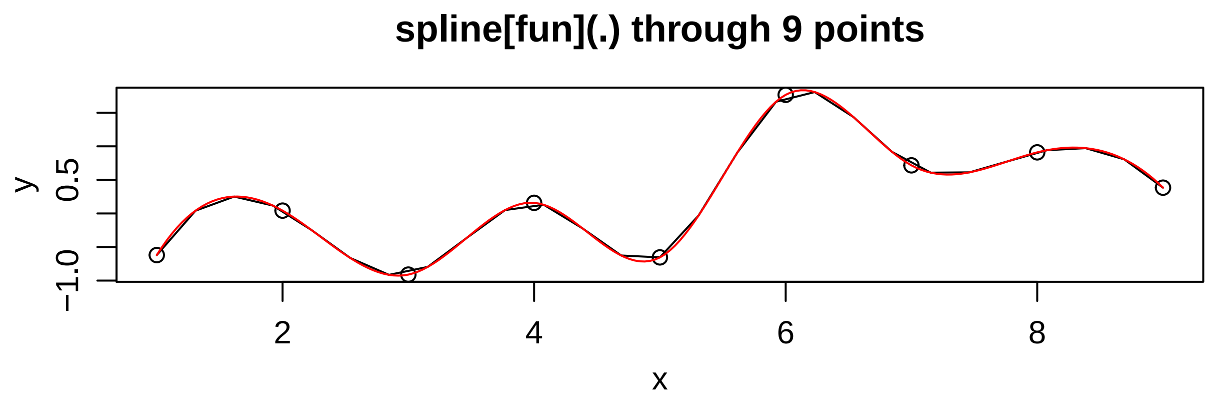
```
dat.te=data.frame(x=Khan$xtest, y=as.factor(Khan$ytest))
pred.te=predict(out, newdata=dat.te)
table(pred.te, dat.te$y)
```

```
##
## pred.te 1 2 3 4
##       1 3 0 0 0
##       2 0 6 2 0
##       3 0 0 4 0
##       4 0 0 0 5
```

15 Exercise 4

15.1 Cubic Spline

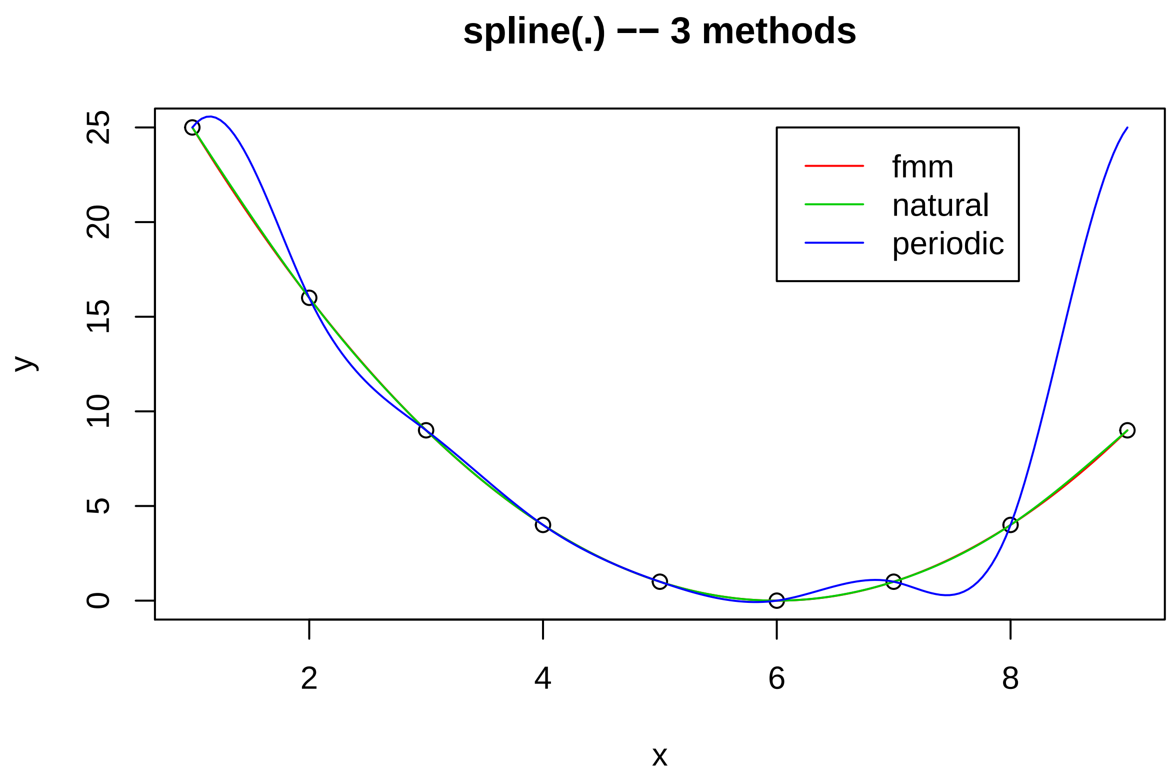
```
op <- par(mfrow = c(2,1), mgp = c(2,.8,0), mar = .1+c(3,3,3,1))
n <- 9
x <- 1:n
y <- rnorm(n)

# Plot
plot(x, y, main = paste("spline[fun](.) through", n, "points"))
lines(spline(x, y))
lines(spline(x, y, n = 201), col = 2)
```



```
y <- (x-6)^2
plot(x, y, main = "spline(.) -- 3 methods")
lines(spline(x, y, n = 201), col = 2)
lines(spline(x, y, n = 201, method = "natural"), col = 3)
lines(spline(x, y, n = 201, method = "periodic"), col = 4)
```

```
## Warning in spline(x, y, n = 201, method = "periodic"): spline: first and
## last y values differ - using y[1] for both
```

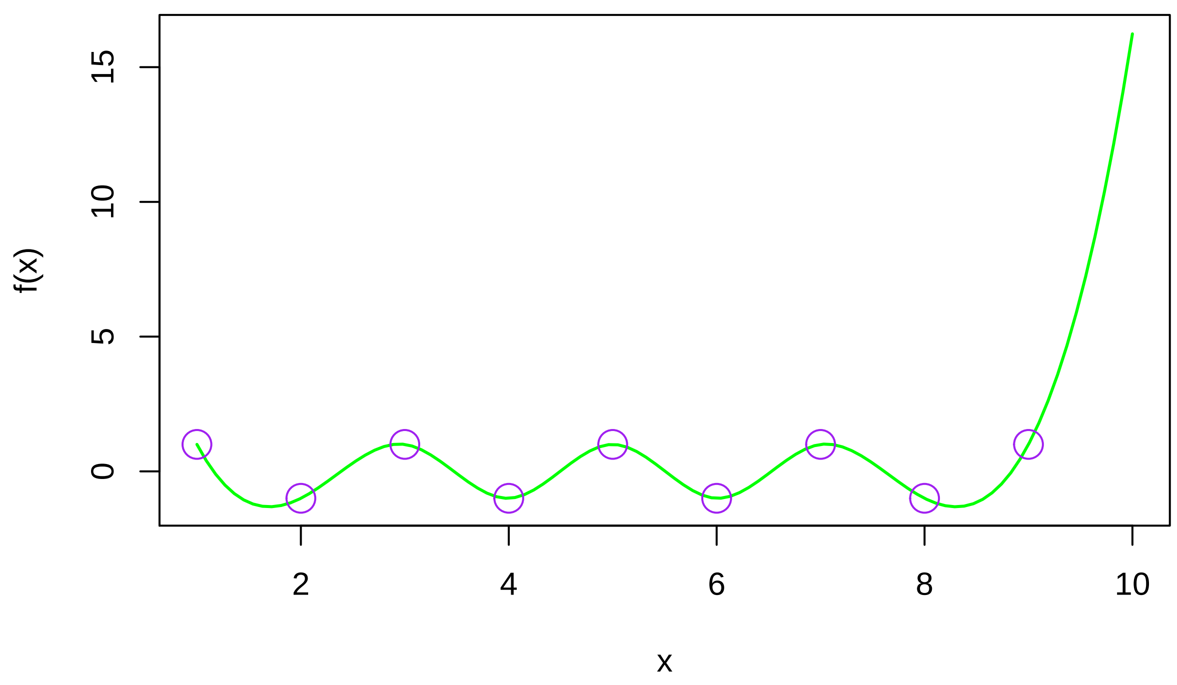
```
legend(6,25, c("fmm","natural","periodic"), col=2:4, lty=1)
```



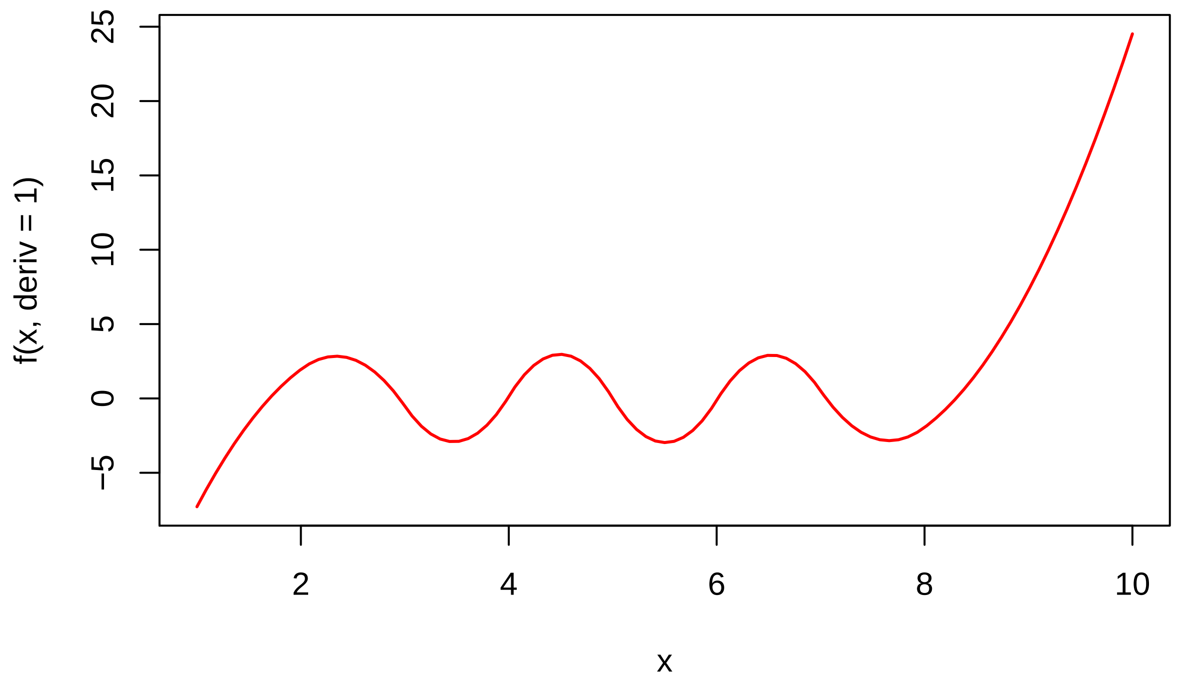
```
y <- sin((x-0.5)*pi)
f <- splinefun(x, y)
ls(envir = environment(f))
```

```
## [1] "z"
```

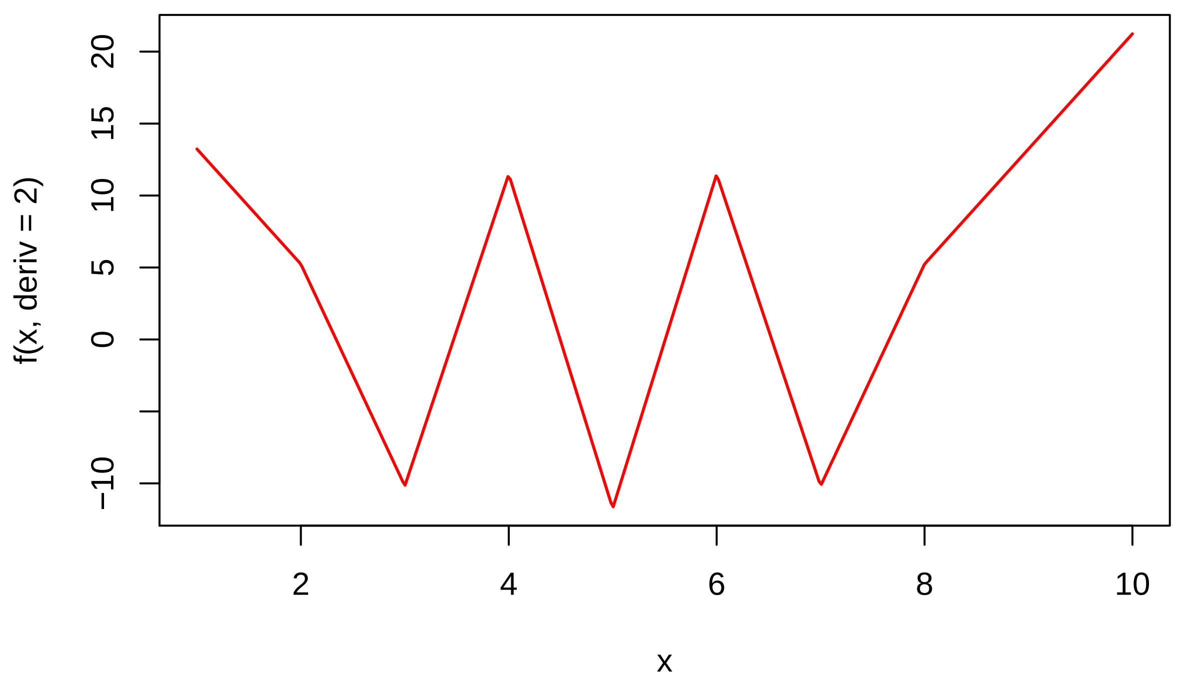
```
splinecoef <- get("z", envir = environment(f))
curve(f(x), 1, 10, col = "green", lwd = 1.5)
points(splinecoef, col = "purple", cex = 2)
```



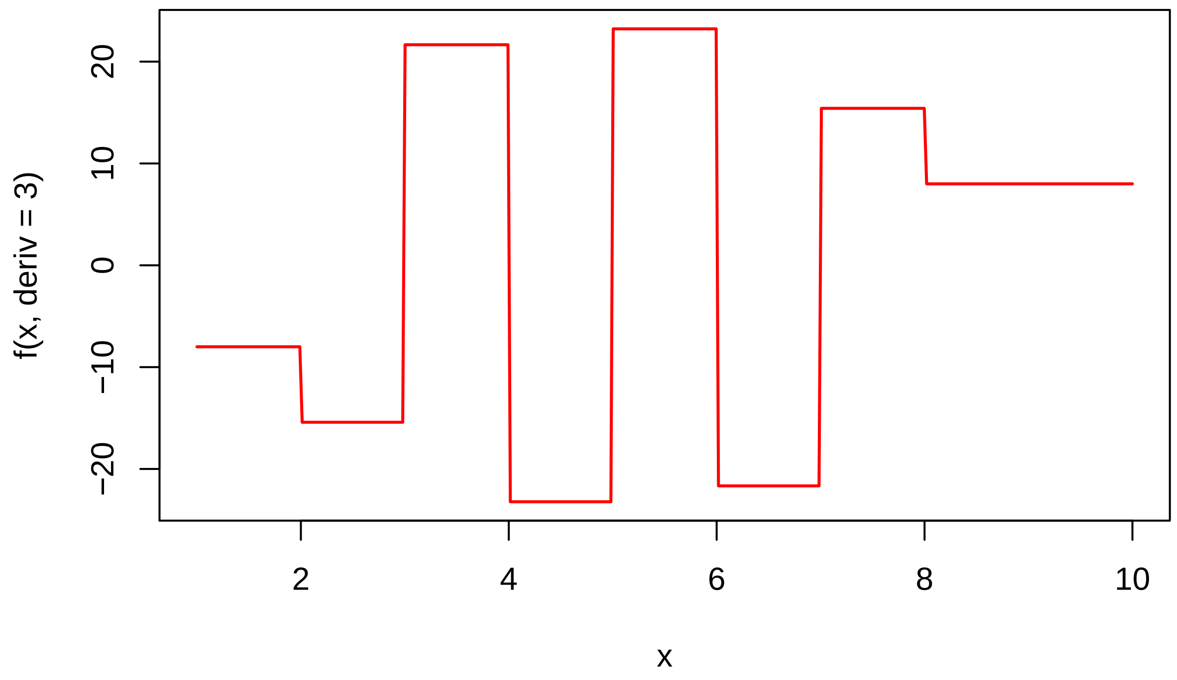
```r
curve(f(x, deriv=1), 1, 10, col = 2, lwd = 1.5)
```



```r
curve(f(x, deriv=2), 1, 10, col = 2, lwd = 1.5, n = 401)
```



```r
curve(f(x, deriv=3), 1, 10, col = 2, lwd = 1.5, n = 401)
```



```r
par(op)
```

15.2 Sampling for Monte Carlo

```r
# Generate a Monte Carlo sample
generateMCSample <- function(n, vals) {
  # Packages to generate quasi-random sequences
  # and rearrange the data
  require(randtoolbox)
  require(plyr)

  # Generate a Sobol' sequence
  sob <- sobol(n, length(vals))

  # Fill a matrix with the values
  # inverted from uniform values to
  # distributions of choice
  samp <- matrix(rep(0,n*(length(vals)+1)), nrow=n)
  samp[,1] <- 1:n
  for (i in 1:length(vals)) {
    l <- vals[[i]]
    dist <- l$dist
    params <- l$params
    samp[,i+1] <- eval(call(paste("q",dist,sep=""),sob[,i],params[1],params[2]))
  }
  # Convert matrix to data frame and label
  samp <- as.data.frame(samp)
  names(samp) <- c("n",laply(vals, function(l) l$var))
  return(samp)
}

# Example:
n <- 1000  # number of simulations to run

# List described the distribution of each variable
vals <- list(list(var="Uniform",
               dist="unif",
               params=c(0,1)),
          list(var="Normal",
               dist="norm",
               params=c(0,1)),
          list(var="Weibull",
               dist="weibull",
               params=c(2,1)))

# Generate the sample
# install.packages('randtoolbox')
library('randtoolbox')
```

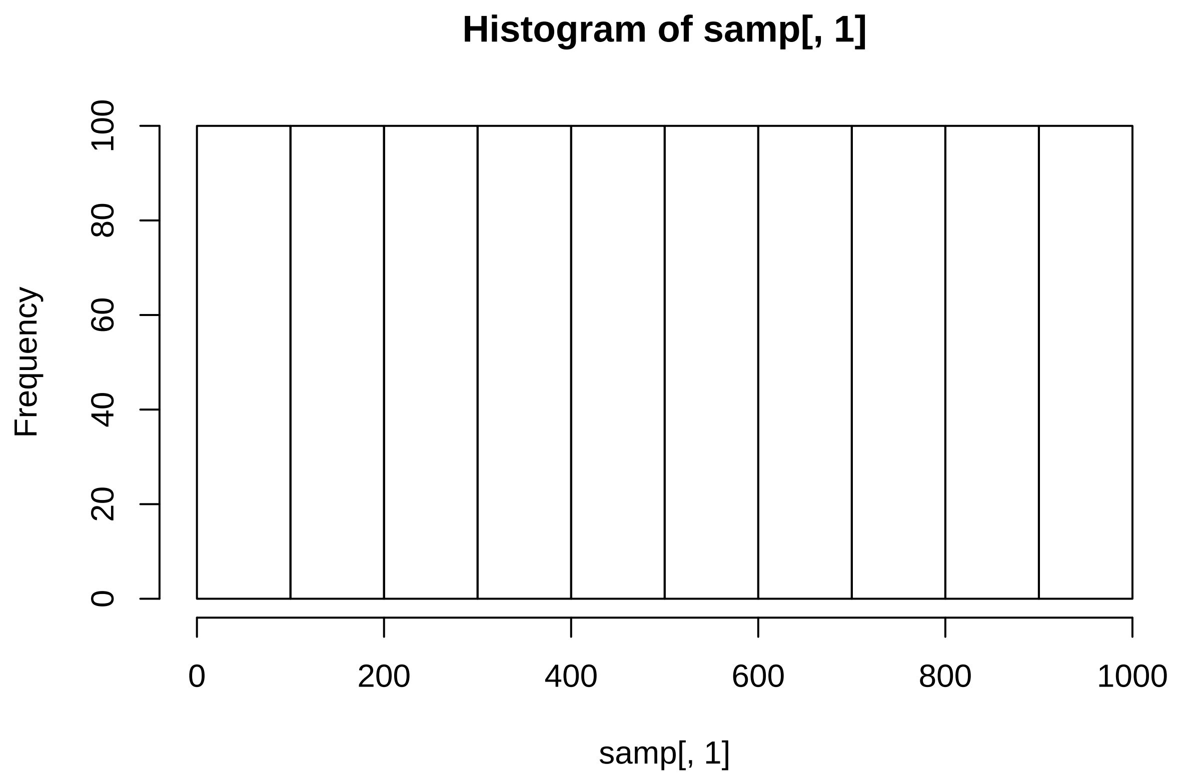
```
## Loading required package: rngWELL
```

```
## This is randtoolbox. For overview, type 'help("randtoolbox")'.
```

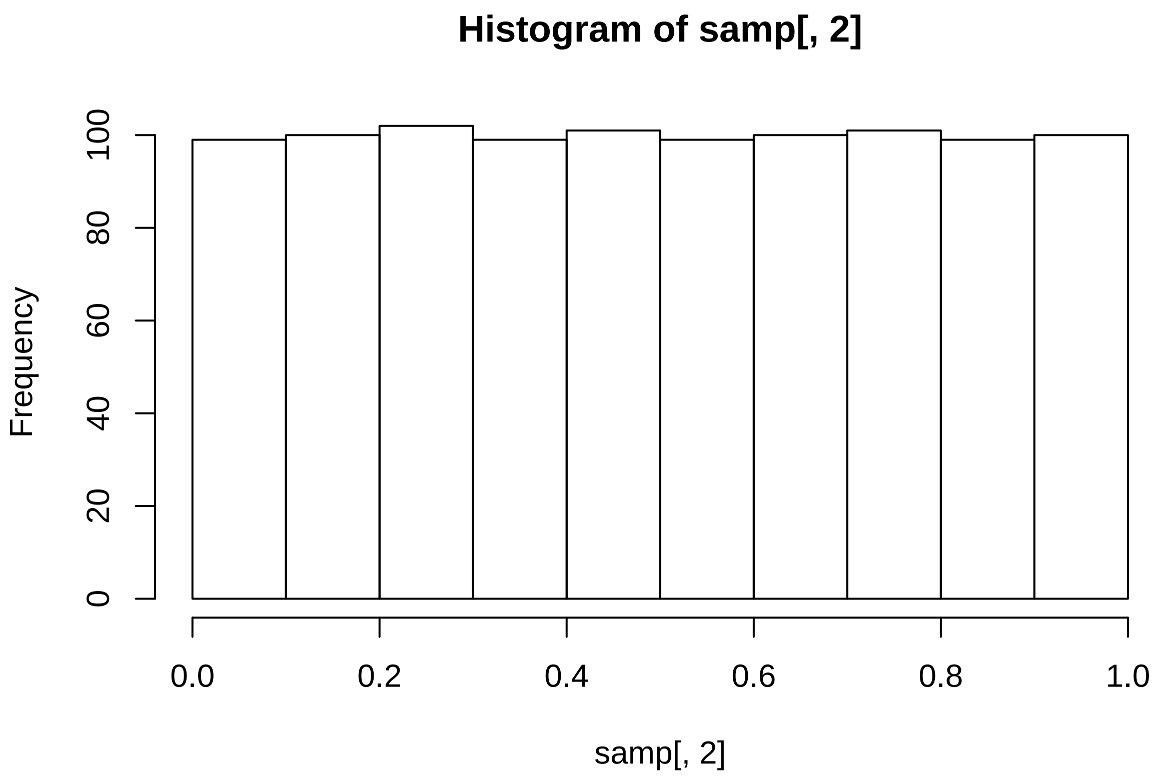
```r
samp <- generateMCSample(n,vals)
```

```
## Loading required package: plyr
```

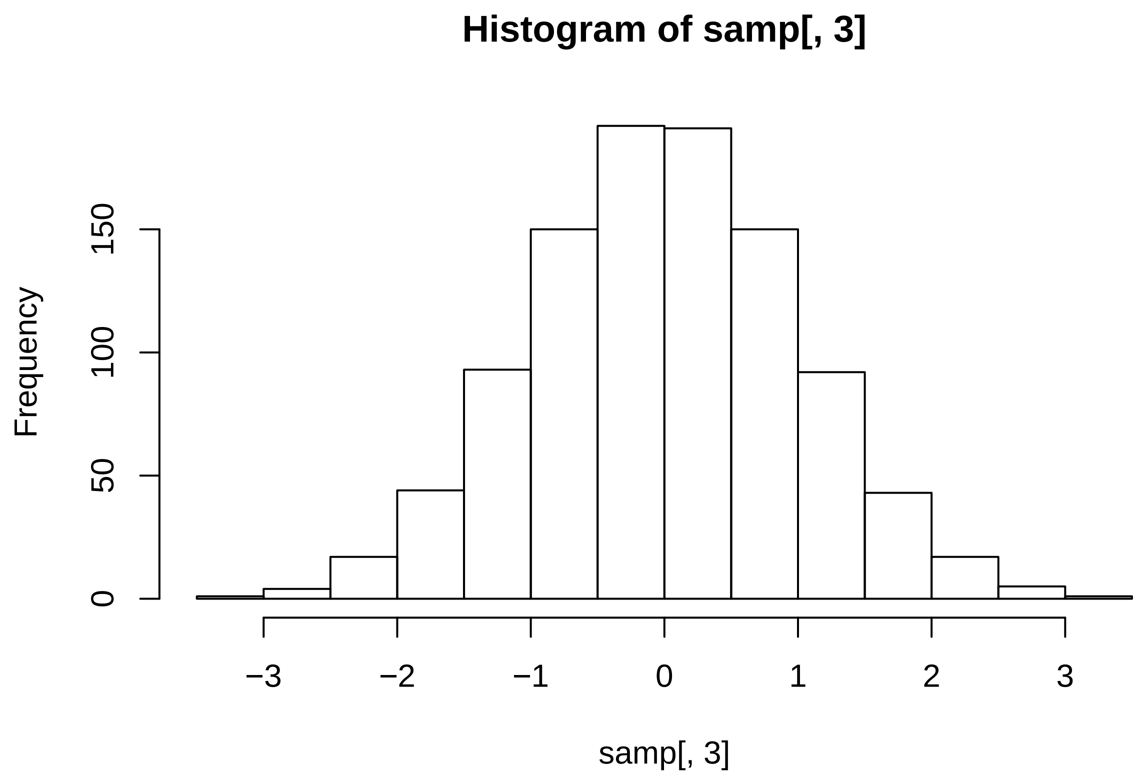
```r
hist(samp[,1])
```


Histogram of samp[, 1]
Frequency
0
20
40
60
80
100
0
200
400
600
800
1000
samp[, 1]

```r
hist(samp[,2])
```


Histogram of samp[, 2]
Frequency
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0
samp[, 2]

```r
hist(samp[,3])
```


Histogram of samp[, 3]
Frequency
0
50
100
150
−3
−2
−1
0
1
2
3
samp[, 3]

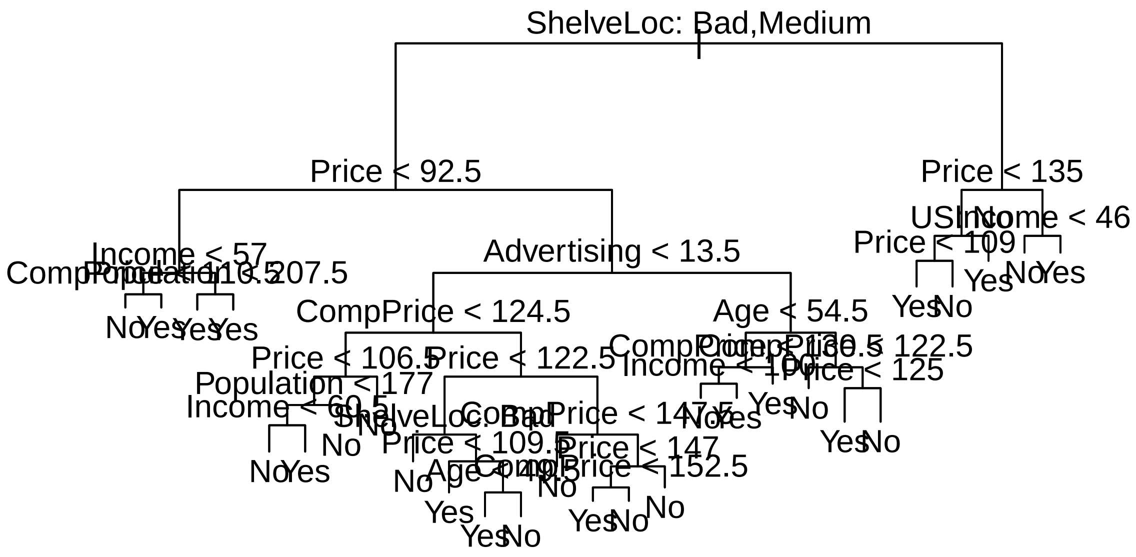
16 Exercise 5

16.1 Fitting Classification Trees

```
# Set up data set:
# install.packages('tree')
library(tree)
library(ISLR)
attach(Carseats)
High <- ifelse(Sales <= 8, "No", "Yes")
Carseats <- data.frame(Carseats, High)

# Fit one classification
# to predict High using all variables but Sales.
tree.carseats <- tree(High~.-Sales,Carseats)
summary(tree.carseats)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

```
# Comment training error is 0.09 = 36/400,
# which is given by equation
# -2 \sum_m \sum_k n_{mk} \log \hat{p}_{mk}
# where n_{mk} is the number of observations
# in the mth terminal node that belong to the kth class.

# Plot:
plot(tree.carseats)
text(tree.carseats, pretty=0)
```



```r
# Each branche:
tree.carseats
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 400 541.500 No ( 0.59000 0.41000 )
##     2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
##       4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
##         8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
##          16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
##          17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
##         9) Income > 57 36  35.470 Yes ( 0.19444 0.80556 )
##          18) Population < 207.5 16  21.170 Yes ( 0.37500 0.62500 ) *
##          19) Population > 207.5 20   7.941 Yes ( 0.05000 0.95000 ) *
##       5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
##        10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
##          20) CompPrice < 124.5 96  44.890 No ( 0.93750 0.06250 )
##            40) Price < 106.5 38  33.150 No ( 0.84211 0.15789 )
##              80) Population < 177 12  16.300 No ( 0.58333 0.41667 )
##               160) Income < 60.5 6   0.000 No ( 1.00000 0.00000 ) *
##               161) Income > 60.5 6   5.407 Yes ( 0.16667 0.83333 ) *
##              81) Population > 177 26   8.477 No ( 0.96154 0.03846 ) *
##            41) Price > 106.5 58   0.000 No ( 1.00000 0.00000 ) *
##          21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
##            42) Price < 122.5 51  70.680 Yes ( 0.49020 0.50980 )
##              84) ShelveLoc: Bad 11   6.702 No ( 0.90909 0.09091 ) *
##               85) ShelveLoc: Medium 40  52.930 Yes ( 0.37500 0.62500 )
##                170) Price < 109.5 16   7.481 Yes ( 0.06250 0.93750 ) *
##                171) Price > 109.5 24  32.600 No ( 0.58333 0.41667 )
##                  342) Age < 49.5 13  16.050 Yes ( 0.30769 0.69231 ) *
##                  343) Age > 49.5 11   6.702 No ( 0.90909 0.09091 ) *
##             43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
##               86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
##               87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
##                174) Price < 147 12  16.300 Yes ( 0.41667 0.58333 )
##                  348) CompPrice < 152.5 7   5.742 Yes ( 0.14286 0.85714 ) *
##                  349) CompPrice > 152.5 5   5.004 No ( 0.80000 0.20000 ) *
##                175) Price > 147 7   0.000 No ( 1.00000 0.00000 ) *
##         11) Advertising > 13.5 45  61.830 Yes ( 0.44444 0.55556 )
##           22) Age < 54.5 25  25.020 Yes ( 0.20000 0.80000 )
##             44) CompPrice < 130.5 14  18.250 Yes ( 0.35714 0.64286 )
##               88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
##               89) Income > 100 5   0.000 Yes ( 0.00000 1.00000 ) *
##             45) CompPrice > 130.5 11   0.000 Yes ( 0.00000 1.00000 ) *
##           23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
##             46) CompPrice < 122.5 10   0.000 No ( 1.00000 0.00000 ) *
##             47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
##               94) Price < 125 5   0.000 Yes ( 0.00000 1.00000 ) *
##               95) Price > 125 5   0.000 No ( 1.00000 0.00000 ) *
##      3) ShelveLoc: Good 85  90.330 Yes ( 0.22353 0.77647 )
##        6) Price < 135 68  49.260 Yes ( 0.11765 0.88235 )
##         12) US: No 17  22.070 Yes ( 0.35294 0.64706 )
##           24) Price < 109 8   0.000 Yes ( 0.00000 1.00000 ) *
##           25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
##         13) US: Yes 51  16.880 Yes ( 0.03922 0.96078 ) *
##        7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
##         14) Income < 46 6   0.000 No ( 1.00000 0.00000 ) *
##         15) Income > 46 11  15.160 Yes ( 0.45455 0.54545 ) *
```

```
# Predict:
set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train,]
High.test <- High[-train]
tree.carseats <- tree(High~.-Sales,Carseats,subset=train)
tree.pred <- predict(tree.carseats,Carseats.test,type="class")
table(tree.pred,High.test)
```

```
##           High.test
## tree.pred No Yes
##       No   86  27
##       Yes 30  57
```

```
# Testing Accuracy:
(86+57)/(86+27+30+57)
```

```
## [1] 0.715
```

```
# Pruning:
# Weather it might lead to improved results:
set.seed(3)
cv.carseats <- cv.tree(tree.carseats,FUN=prune.misclass)
```

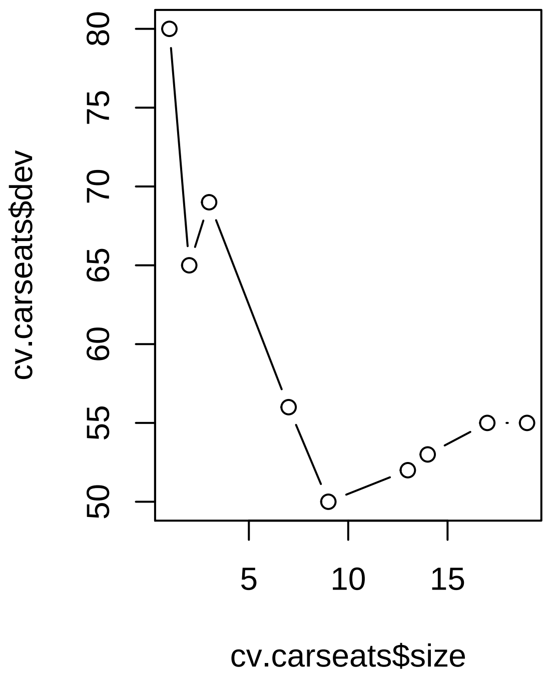
```r
names(cv.carseats)
```

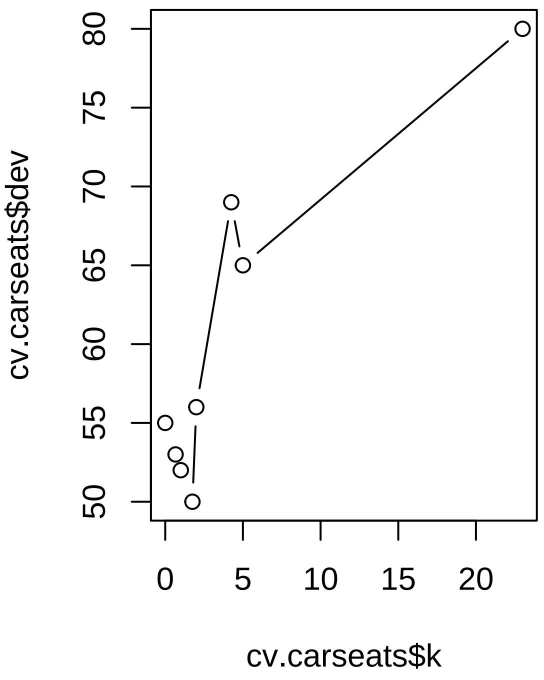
```
## [1] "size"   "dev"    "k"      "method"
```

```r
cv.carseats
```

```
## $size
## [1] 19 17 14 13  9  7  3  2  1
##
## $dev
## [1] 55 55 53 52 50 56 69 65 80
##
## $k
## [1]       -Inf  0.0000000  0.6666667  1.0000000  1.7500000  2.0000000
## [7]  4.2500000  5.0000000 23.0000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
# Comment:
# Function cv.tree() performs cross-validation to
# determine the optimal level of tree complexity
# cost complexity pruning is used in order to select
# a sequence of trees for consideration.
# We use the argument FUN=prune.misclass in order
# to indicate that we want the classification
# error rate to guide the cross-validation
# and pruning process, rather than the default
# for the cv.treee() function, which is
# deviance.

# Plot the erroras a function of size and k:
par(mfrow=c(1,2))
plot(cv.carseats$size,cv.carseats$dev,type="b")
plot(cv.carseats$k,cv.carseats$dev,type="b")
```





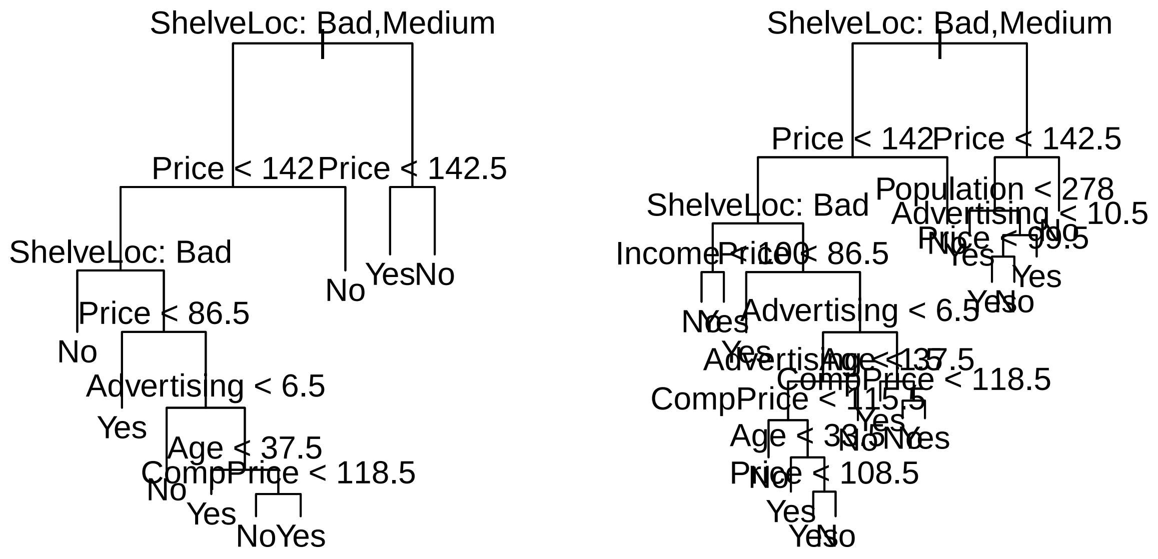
```
# Pruning:
prune.carseats <- prune.misclass(tree.carseats, best=9)
plot(prune.carseats)
text(prune.carseats,pretty=0)

# Predict:
tree.pred <- predict(prune.carseats,Carseats.test,type="class")
table(tree.pred,High.test)
```

```
##          High.test
## tree.pred No Yes
##       No  94  24
##       Yes 22  60
```

```
sum(diag(table(tree.pred,High.test)))/sum(table(tree.pred,High.test))
```

```
## [1] 0.77
```

```
# Increased best, would give lower classification accuracy:
prune.carseats <- prune.misclass(tree.carseats, best=15)
plot(prune.carseats)
text(prune.carseats,pretty=0)
```



```
tree.pred <- predict(prune.carseats,Carseats.test,type="class")
table(tree.pred,High.test)
```

```
##          High.test
## tree.pred No Yes
##       No  86  22
##       Yes 30  62
```

```
sum(diag(table(tree.pred,High.test)))/sum(table(tree.pred,High.test))
```

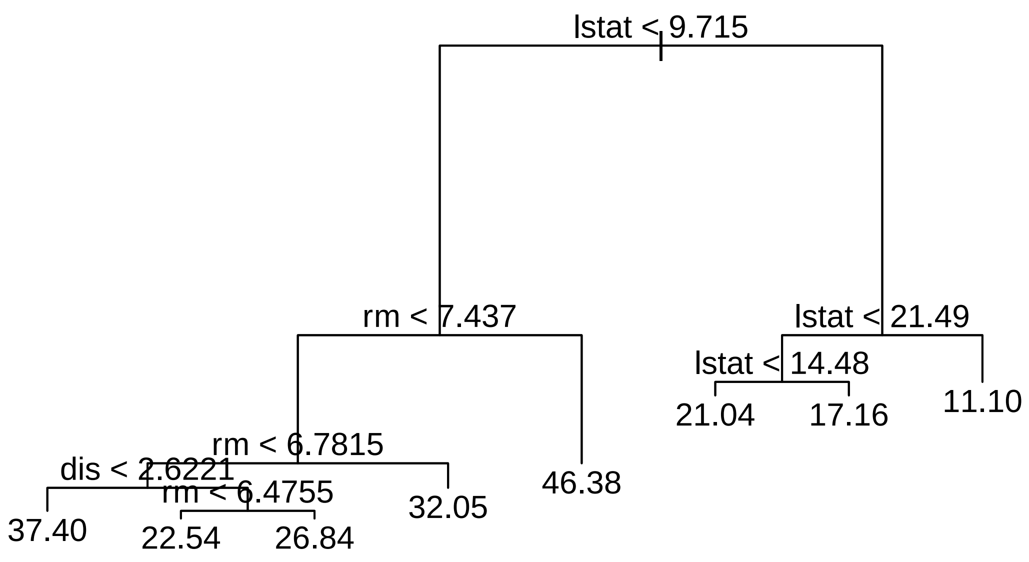
```
## [1] 0.74
```

16.2 Fitting Regression Trees

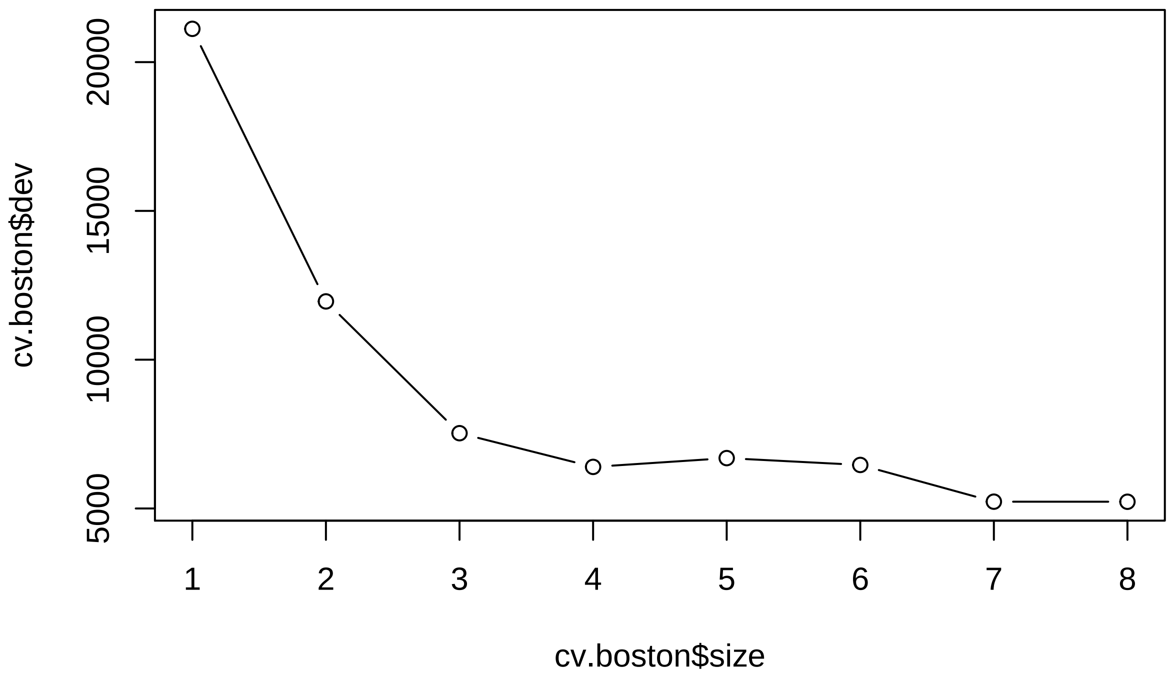
```
library(MASS)
set.seed(1)
train <- sample(1:nrow(Boston),nrow(Boston)/2)
tree.boston <- tree(medv~.,Boston,subset=train)
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8
## Residual mean deviance:  12.65 = 3099 / 245
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -14.10000  -2.04200  -0.05357   0.00000   1.96000  12.60000
```

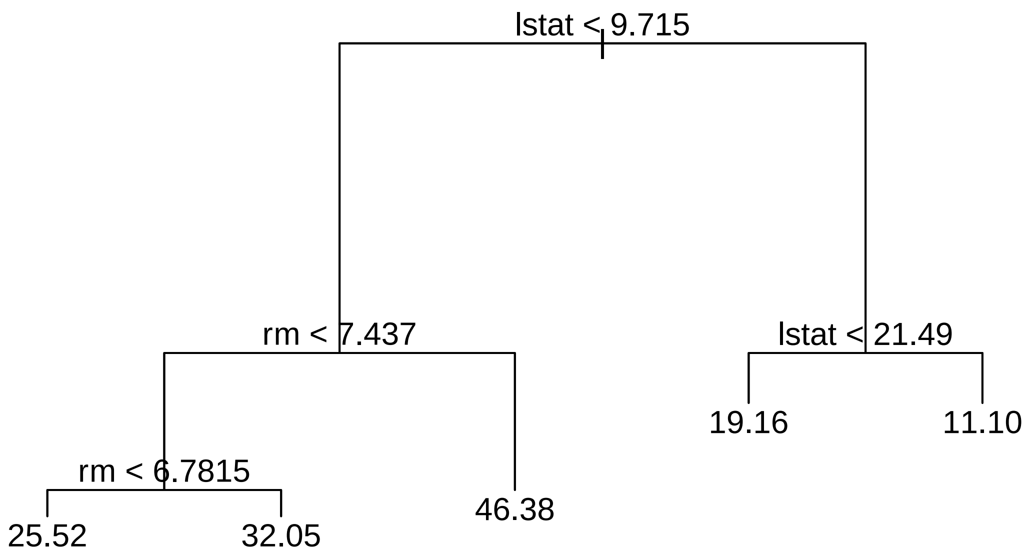
```
# Plot
plot(tree.boston)
text(tree.boston,pretty=0)
```



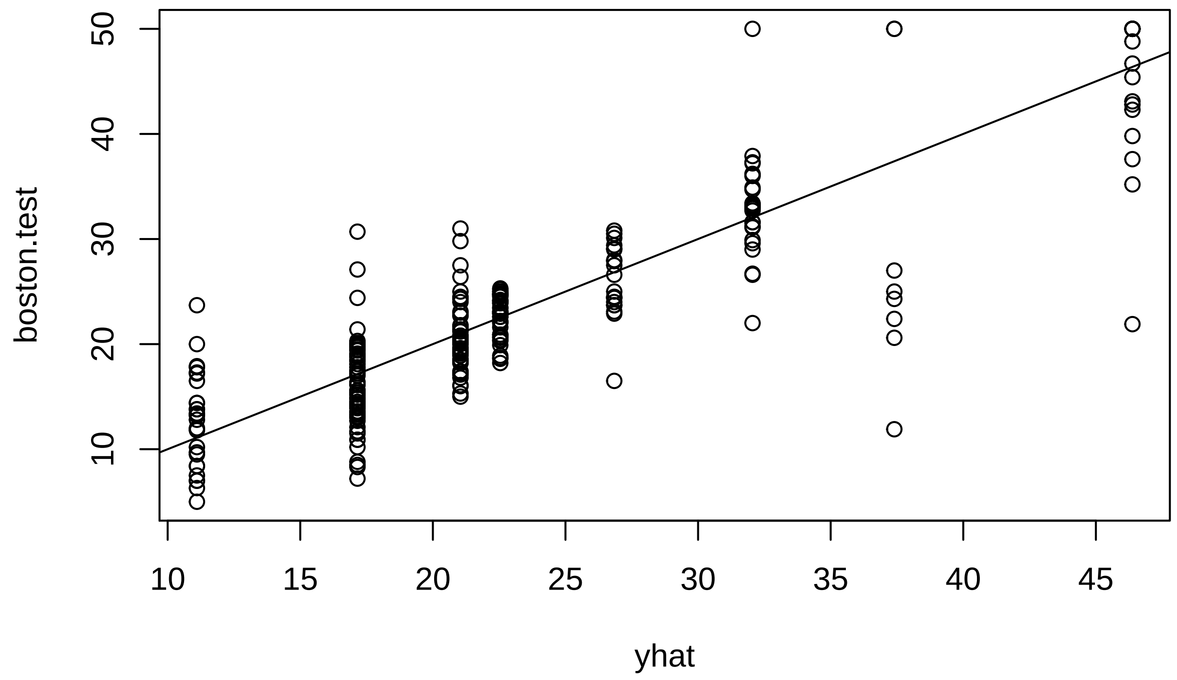
```
# Use cv.tree() function to see whether
# pruning the tree will improve performance
cv.boston <- cv.tree(tree.boston)
plot(cv.boston$size,cv.boston$dev,type='b')
```



```r
# Pruning
prune.boston <- prune.tree(tree.boston,best=5)
plot(prune.boston)
text(prune.boston,pretty=0)
```



```
# Prediction:
yhat <- predict(tree.boston,newdata=Boston[-train,])
boston.test <- Boston[-train,"medv"]
plot(yhat,boston.test)
abline(0,1)
```



```
mean((yhat-boston.test)^2)
```

```
## [1] 25.04559
```

16.3 Bagging and Random Forests

```
# Here we apply bagging and random forests to
# the Boston data, using the randomForest
# package in R.

# Package:
library(randomForest)
```

```
## randomForest 4.6-12
```

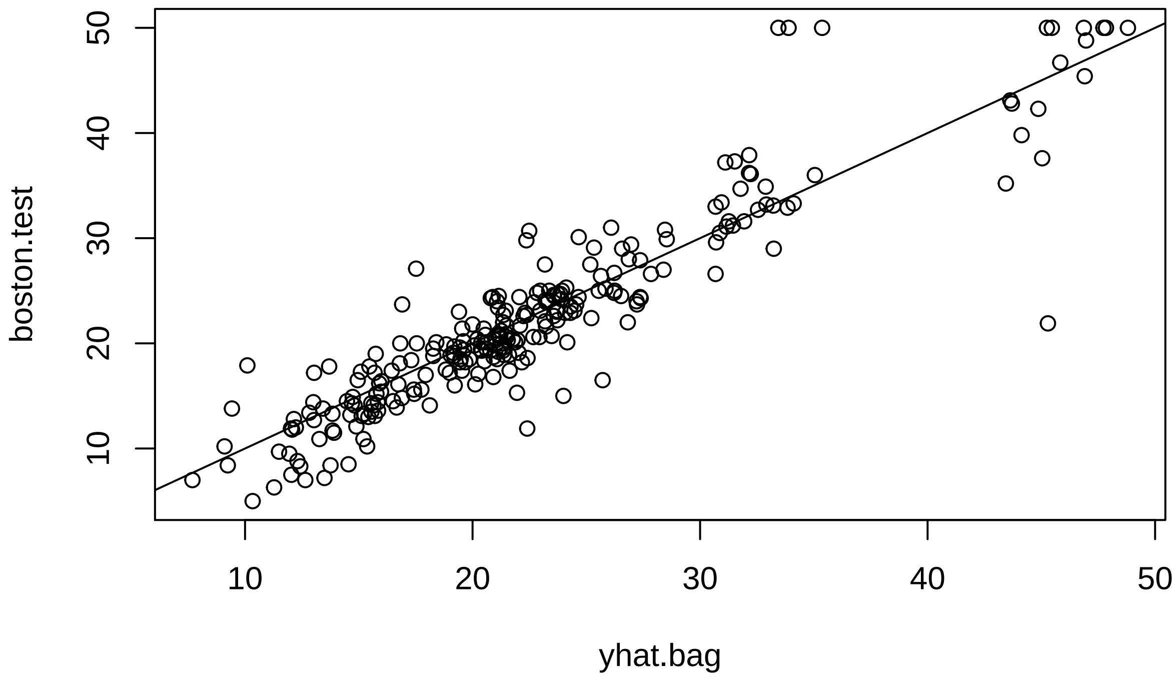
```
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(1)

# Random Forest
bag.boston <- randomForest(medv~.,
                           data=Boston,subset=train,
                           mtry=13,importance=TRUE)
bag.boston
```

```
##
## Call:
```

```
##  randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 11.02509
##                     % Var explained: 86.65
```

```
# Comment:
# mtry=13 indicates that all 13 predictors should be considered
# for each split of the tree. That is, that bagging should
# be done.

# How well does it perform?
yhat.bag <- predict(bag.boston,newdata=Boston[-train,])
plot(yhat.bag, boston.test)
abline(0,1)
```



```
mean((yhat.bag - boston.test)^2)
```

```
## [1] 13.47349
```

```
# MSE is a lot smaller.

# Change randomForest() using the ntree argument:
bag.boston <- randomForest(medv~.,data=Boston,subset=train,
                           mtry=13,ntree=25)
yhat.bag <- predict(bag.boston,newdata=Boston[-train,])
```

```
mean((yhat.bag - boston.test)^2)
```

```
## [1] 13.43068
```

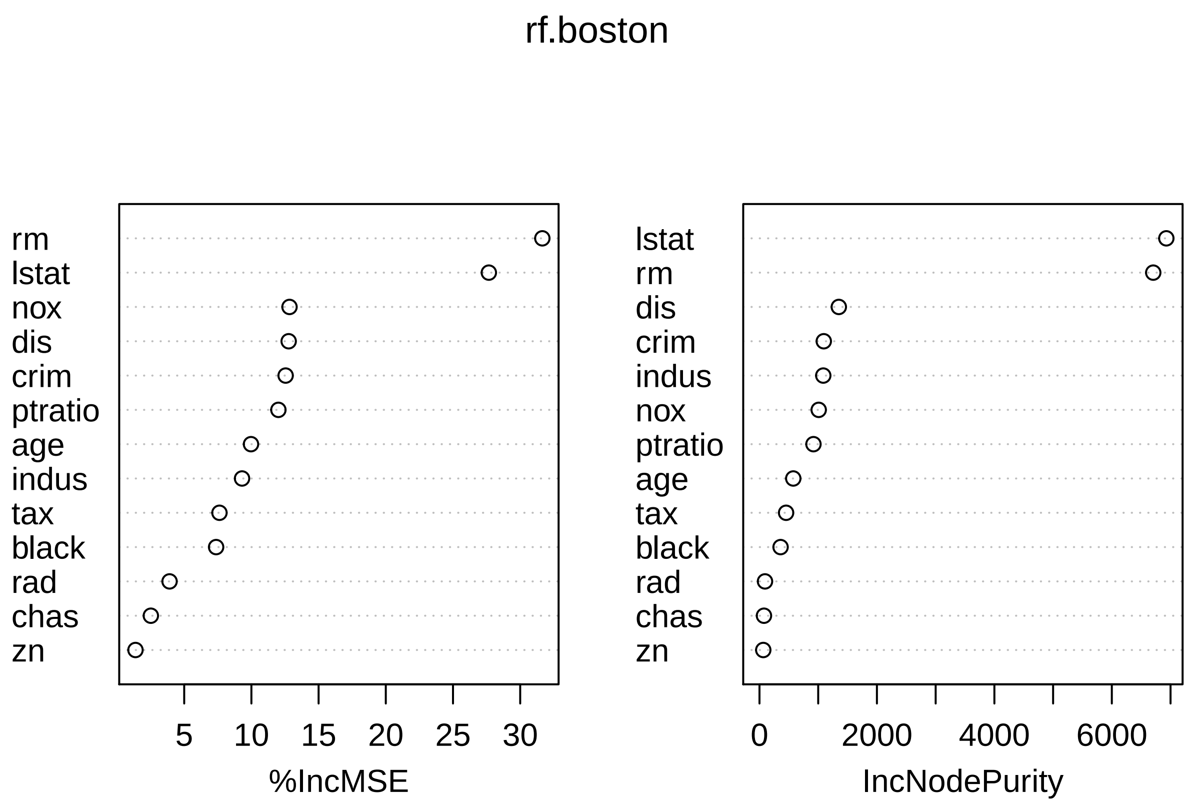
```
# Try mtry = 6:
set.seed(1)
rf.boston <- randomForest(medv~.,data=Boston,subset=train,
                          mtry=6,importance=TRUE)
yhat.rf <- predict(rf.boston,newdata=Boston[-train,])
mean((yhat.rf - boston.test)^2)
```

```
## [1] 11.48022
```

```
# We have another improvement.

# Using importance() function to see
# importance of each variable.
importance(rf.boston)
```

```
##           %IncMSE IncNodePurity
## crim    12.547772    1094.65382
## zn       1.375489      64.40060
## indus    9.304258    1086.09103
## chas     2.518766      76.36804
## nox     12.835614    1008.73703
## rm      31.646147    6705.02638
## age      9.970243     575.13702
## dis     12.774430    1351.01978
## rad      3.911852      93.78200
## tax      7.624043     453.19472
## ptratio 12.008194     919.06760
## black    7.376024     358.96935
## lstat   27.666896    6927.98475
```

```
# Plot these importance measures:
varImpPlot(rf.boston)
```

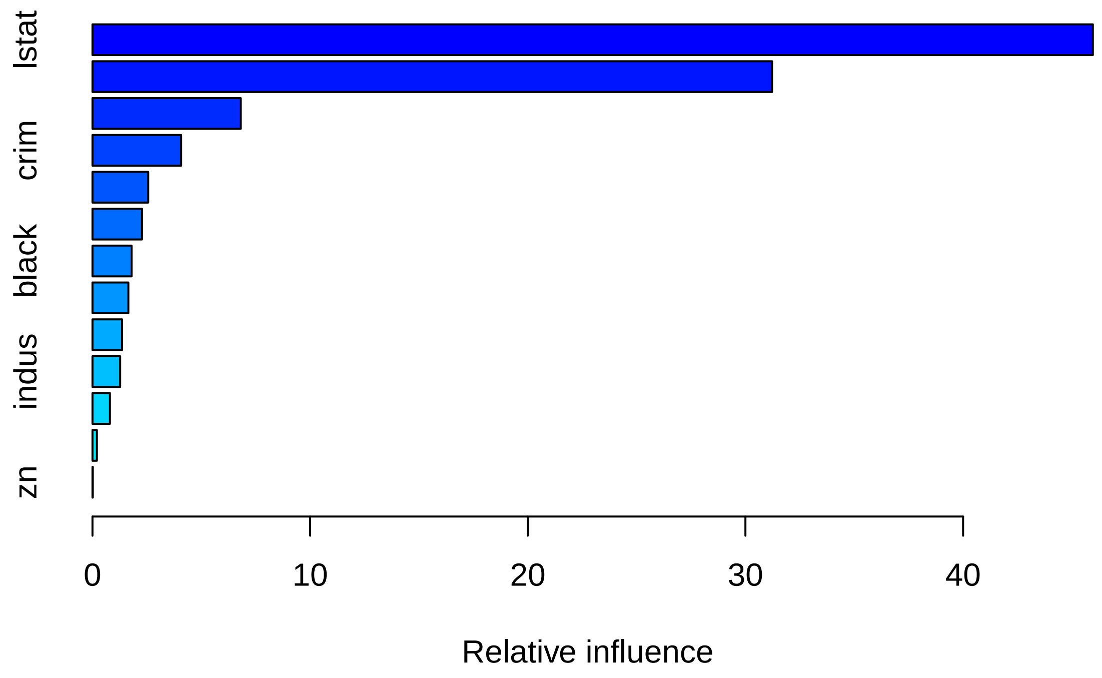


16.4 Boosting

```
# We use gbm package
library(gbm)
```

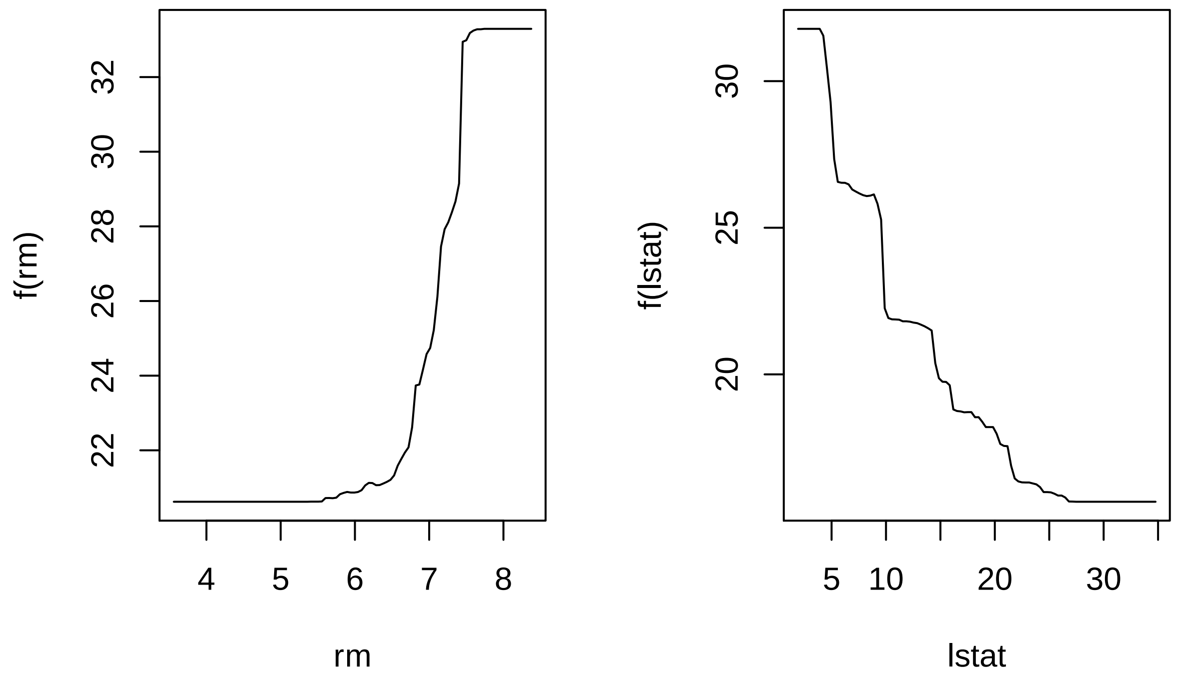
```
## Loading required package: survival

## Loading required package: lattice

## Loading required package: parallel

## Loaded gbm 2.1.3
```

```
set.seed(1)
boost.boston <- gbm(medv~.,data=Boston[train,],
                    distribution="gaussian",
                    n.trees=5000,interaction.depth=4)
summary(boost.boston)
```



```
##             var    rel.inf
## lstat     lstat 45.9627334
## rm           rm 31.2238187
## dis         dis  6.8087398
## crim       crim  4.0743784
## nox         nox  2.5605001
## ptratio ptratio  2.2748652
## black     black  1.7971159
## age         age  1.6488532
## tax         tax  1.3595005
## indus     indus  1.2705924
## chas       chas  0.8014323
## rad         rad  0.2026619
## zn           zn  0.0148083
```

```
# We see that lstat and rm are by far the most
# important variables.
# We can also produce partial dependence plots for
# these two variables. These plots illustrate the
# marginal effect of the selected variables
# on the response after integrating out the other variables.

par(mfrow=c(1,2))
plot(boost.boston,i="rm")
plot(boost.boston,i="lstat")
```



```
# Test set:
yhat.boost <- predict(boost.boston, newdata=Boston[-train,],
                      n.trees=5000)
mean((yhat.boost - boston.test)^2)
```

```
## [1] 11.84434
```

```
# How to improve?
# We can perform boosting with a different value of the shrinkage
# parameter lambda . The default value is 0.001.
boost.boston <- gbm(medv~.,data=Boston[train,],
                    distribution="gaussian",n.trees=5000,
                    interaction.depth=4,shrinkage=0.2,
                    verbose=F)
yhat.boost <- predict(boost.boston,newdata=Boston[-train,],
                      n.trees=5000)
mean((yhat.boost - boston.test)^2)
```

```
## [1] 11.51109
```

```
# This change, lambda=0.2, leads to a slightly lower MSE.
```

17 Exercise 6

17.1 Neural Network

```
# Source:
# https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

# Abstract:
# Features are computed from a digitized image of a fine needle
# aspirate (FNA) of a breast mass. They describe characteristics
# of the cell nuclei present in the image. n the 3-dimensional
# space is that described in:
# [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming
# Discrimination of Two Linearly Inseparable Sets", Optimization Methods
# and Software 1, 1992, 23-34].

# Attribute Information:
# 1) ID number
# 2) Diagnosis (M = malignant, B = benign) 3-32)
# Ten real-valued features are computed for each cell nucleus:
# a) radius (mean of distances from center to points on the
# perimeter)
# b) texture (standard deviation of gray-scale values)
# c) perimeter
# d) area
# e) smoothness (local variation in radius lengths)
# f) compactness (perimeter^2 / area - 1.0)
# g) concavity (severity
# of concave portions of the contour)
# h) concave points (number of concave portions of the
# contour) i) symmetry j) fractal dimension
# ("coastline approximation" -
# 1) The mean, standard error and "worst" or largest (mean of the three
# largest values) of these features were computed for each image,
# resulting in 30 features. For instance, field 3 is Mean Radius,
# field 13 is Radius SE, field 23 is Worst Radius.
# All feature values are recoded with four significant digits.
# Missing attribute values: none
# Class distribution: 357 benign, 212 malignant

################# LOADING DATA ###########################

library('mxnet')
```

```
## Init Rcpp
```

```
# Load data:
all <- read.csv('F:/data_b_cancer/data.csv', header = TRUE)
all <- all[,-1] # Get rid of ID
colnames(all)[1] <- "Diagnosis"; head(all); dim(all); names(all)
```

```
##   Diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1         M       17.99        10.38         122.80    1001.0
## 2         M       20.57        17.77         132.90    1326.0
## 3         M       19.69        21.25         130.00    1203.0
## 4         M       11.42        20.38          77.58     386.1
## 5         M       20.29        14.34         135.10    1297.0
## 6         M       12.45        15.70          82.57     477.1
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1         0.11840          0.27760         0.3001             0.14710
## 2         0.08474          0.07864         0.0869             0.07017
## 3         0.10960          0.15990         0.1974             0.12790
## 4         0.14250          0.28390         0.2414             0.10520
## 5         0.10030          0.13280         0.1980             0.10430
## 6         0.12780          0.17000         0.1578             0.08089
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1        0.2419                0.07871    1.0950     0.9053        8.589
## 2        0.1812                0.05667    0.5435     0.7339        3.398
## 3        0.2069                0.05999    0.7456     0.7869        4.585
## 4        0.2597                0.09744    0.4956     1.1560        3.445
## 5        0.1809                0.05883    0.7572     0.7813        5.438
## 6        0.2087                0.07613    0.3345     0.8902        2.217
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1  153.40      0.006399        0.04904      0.05373           0.01587
## 2   74.08      0.005225        0.01308      0.01860           0.01340
## 3   94.03      0.006150        0.04006      0.03832           0.02058
## 4   27.23      0.009110        0.07458      0.05661           0.01867
## 5   94.44      0.011490        0.02461      0.05688           0.01885
## 6   27.19      0.007510        0.03345      0.03672           0.01137
##   symmetry_se fractal_dimension_se radius_worst texture_worst
## 1     0.03003             0.006193        25.38         17.33
## 2     0.01389             0.003532        24.99         23.41
## 3     0.02250             0.004571        23.57         25.53
## 4     0.05963             0.009208        14.91         26.50
## 5     0.01756             0.005115        22.54         16.67
## 6     0.02165             0.005082        15.47         23.75
##   perimeter_worst area_worst smoothness_worst compactness_worst
## 1          184.60     2019.0           0.1622            0.6656
## 2          158.80     1956.0           0.1238            0.1866
## 3          152.50     1709.0           0.1444            0.4245
## 4           98.87      567.7           0.2098            0.8663
## 5          152.20     1575.0           0.1374            0.2050
## 6          103.40      741.6           0.1791            0.5249
##   concavity_worst concave.points_worst symmetry_worst
## 1          0.7119               0.2654         0.4601
## 2          0.2416               0.1860         0.2750
## 3          0.4504               0.2430         0.3613
## 4          0.6869               0.2575         0.6638
## 5          0.4000               0.1625         0.2364
## 6          0.5355               0.1741         0.3985
##   fractal_dimension_worst
## 1                 0.11890
## 2                 0.08902
## 3                 0.08758
## 4                 0.17300
## 5                 0.07678
## 6                 0.12440
## [1] 569  31
```

```
##  [1] "Diagnosis"               "radius_mean"
##  [3] "texture_mean"            "perimeter_mean"
##  [5] "area_mean"               "smoothness_mean"
##  [7] "compactness_mean"        "concavity_mean"
##  [9] "concave.points_mean"     "symmetry_mean"
## [11] "fractal_dimension_mean"  "radius_se"
## [13] "texture_se"              "perimeter_se"
## [15] "area_se"                 "smoothness_se"
## [17] "compactness_se"          "concavity_se"
## [19] "concave.points_se"       "symmetry_se"
## [21] "fractal_dimension_se"    "radius_worst"
## [23] "texture_worst"           "perimeter_worst"
## [25] "area_worst"              "smoothness_worst"
## [27] "compactness_worst"       "concavity_worst"
## [29] "concave.points_worst"    "symmetry_worst"
## [31] "fractal_dimension_worst"
```

```
# Create Dummies:
all$Diagnosis <- ifelse(all$Diagnosis == "M", 1, 0)
head(all); dim(all); names(all) # Check!
```

```
##   Diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1         1       17.99        10.38         122.80    1001.0
## 2         1       20.57        17.77         132.90    1326.0
## 3         1       19.69        21.25         130.00    1203.0
## 4         1       11.42        20.38          77.58     386.1
## 5         1       20.29        14.34         135.10    1297.0
## 6         1       12.45        15.70          82.57     477.1
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1         0.11840          0.27760         0.3001             0.14710
## 2         0.08474          0.07864         0.0869             0.07017
## 3         0.10960          0.15990         0.1974             0.12790
## 4         0.14250          0.28390         0.2414             0.10520
## 5         0.10030          0.13280         0.1980             0.10430
## 6         0.12780          0.17000         0.1578             0.08089
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1        0.2419                0.07871    1.0950     0.9053        8.589
## 2        0.1812                0.05667    0.5435     0.7339        3.398
## 3        0.2069                0.05999    0.7456     0.7869        4.585
## 4        0.2597                0.09744    0.4956     1.1560        3.445
## 5        0.1809                0.05883    0.7572     0.7813        5.438
## 6        0.2087                0.07613    0.3345     0.8902        2.217
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1  153.40      0.006399        0.04904      0.05373           0.01587
## 2   74.08      0.005225        0.01308      0.01860           0.01340
## 3   94.03      0.006150        0.04006      0.03832           0.02058
## 4   27.23      0.009110        0.07458      0.05661           0.01867
## 5   94.44      0.011490        0.02461      0.05688           0.01885
## 6   27.19      0.007510        0.03345      0.03672           0.01137
##   symmetry_se fractal_dimension_se radius_worst texture_worst
## 1     0.03003             0.006193        25.38         17.33
## 2     0.01389             0.003532        24.99         23.41
## 3     0.02250             0.004571        23.57         25.53
## 4     0.05963             0.009208        14.91          26.50
## 5     0.01756             0.005115        22.54          16.67
## 6     0.02165             0.005082        15.47          23.75
##   perimeter_worst area_worst smoothness_worst compactness_worst
## 1          184.60     2019.0           0.1622            0.6656
## 2          158.80     1956.0           0.1238            0.1866
## 3          152.50     1709.0           0.1444            0.4245
## 4           98.87      567.7           0.2098            0.8663
## 5          152.20     1575.0           0.1374            0.2050
## 6          103.40      741.6           0.1791            0.5249
##   concavity_worst concave.points_worst symmetry_worst
## 1          0.7119               0.2654         0.4601
## 2          0.2416               0.1860         0.2750
## 3          0.4504               0.2430         0.3613
## 4          0.6869               0.2575         0.6638
## 5          0.4000               0.1625         0.2364
## 6          0.5355               0.1741         0.3985
##   fractal_dimension_worst
## 1                 0.11890
## 2                 0.08902
## 3                 0.08758
## 4                 0.17300
## 5                 0.07678
## 6                 0.12440

## [1] 569  31

##  [1] "Diagnosis"               "radius_mean"
##  [3] "texture_mean"            "perimeter_mean"
##  [5] "area_mean"               "smoothness_mean"
##  [7] "compactness_mean"        "concavity_mean"
##  [9] "concave.points_mean"     "symmetry_mean"
## [11] "fractal_dimension_mean"  "radius_se"
## [13] "texture_se"              "perimeter_se"
## [15] "area_se"                 "smoothness_se"
## [17] "compactness_se"          "concavity_se"
## [19] "concave.points_se"       "symmetry_se"
## [21] "fractal_dimension_se"    "radius_worst"
## [23] "texture_worst"           "perimeter_worst"
## [25] "area_worst"              "smoothness_worst"
## [27] "compactness_worst"       "concavity_worst"
## [29] "concave.points_worst"    "symmetry_worst"
## [31] "fractal_dimension_worst"
```

```
# Shuffle:
#all <- all[sample(nrow(all), nrow(all)), ]
#head(all); dim(all); names(all)

################# NEURO NETWORK ##########################

# All entries take 0 and 1:
all.copy <- all
for (i in 1:nrow(all.copy)){
  for (j in 1:ncol(all.copy)){
    all.copy[i,j] <- ifelse(all.copy[i,j] <= mean(all.copy[,j]),0,1)
  }
  #print(cbind("Done with", i))
}; head(all.copy); all <- all.copy
```

```
##   Diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1         1           1            0              1         1
## 2         1           1            0              1         1
## 3         1           1            1              1         1
## 4         1           0            1              0         0
## 5         1           1            0              1         1
## 6         1           0            0              0         0
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1               1                1              1                   1
## 2               0                0              0                   1
## 3               1                1              1                   1
## 4               1                1              1                   1
## 5               0                1              1                   1
## 6               1                1              1                   1
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1             1                      1         1          0            1
## 2             0                      0         1          0            1
## 3             1                      0         1          0            1
## 4             1                      1         1          0            1
## 5             0                      0         1          0            1
## 6             1                      1         0          0            0
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1       1             0              1            1                 1
## 2       1             0              0            0                 0
## 3       1             0              1            1                 1
## 4       0             1              1            1                 1
## 5       1             1              0            1                 1
## 6       0             0              1            0                 0
##   symmetry_se fractal_dimension_se radius_worst texture_worst
## 1           1                    1            1             0
## 2           0                    0            1             0
## 3           1                    0            1             0
## 4           1                    1            0             1
## 5           0                    0            1             0
## 6           0                    0            0             0
##   perimeter_worst area_worst smoothness_worst compactness_worst
## 1               1          1                1                 1
## 2               1          1                0                 0
## 3               1          1                1                 1
## 4               0          0                1                 1
## 5               1          1                1                 0
## 6               0          0                1                 1
##   concavity_worst concave.points_worst symmetry_worst
## 1               1                    1              1
## 2               0                    1              0
## 3               1                    1              1
## 4               1                    1              1
## 5               1                    1              0
## 6               1                    1              1
##   fractal_dimension_worst
## 1                        1
## 2                        1
## 3                        1
## 4                        1
## 5                        0
## 6                        1
```

```
# Replace NA entry with 0:
# all[is.na(all)] <- 0; head(all); dim(all)

# Split:
train <- all[1:(0.8*nrow(all)),]; dim(train) # Training set
```

```
## [1] 455  31
```

```
test <- all[(0.8*nrow(all)+1):nrow(all),]; dim(test) # Testing set
```

```
## [1] 113  31
```

```
# Identify Response and Explanatory:
train.x <- train[,-1]; dim(train.x)
```

```
## [1] 455  30
```

```
train.y <- train[,1]; head(train.y)
```

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

```
test.x <- test[,-1]; dim(test.x)
```

```
## [1] 113  30
```

```
test.y <- test[,1]; dim(data.frame(test.y))
```

```
## [1] 113   1
```

```
# Transpose:
train.x <- t(train.x)
test.x <- t(test.x)

# Parameters:
a1 <- 128+4*256 # LeCun: 128
a2 <- 64+4*128 # LeCun: 64
a3 <- 32+4*0 # LeCun: 10
a4 <- 10
iter <- 30

# Configure Network:

data <- mx.symbol.Variable("data")
# In mxnet, we use the data type symbol
# to configure the network.
# data <- mx.symbol.Variable("data")
# uses data to represent the input
# data, i.e., the input layer.
fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=a1) # 1:128; 2:300; 3:400
# We set the first hidden
# layer with fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128).
# This layer has data as the input,
```

```r
# its name, and the number of hidden neurons.
act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu")
# Activation is set with
# act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu").
# The activation function takes the output from the first hidden layer, fc1.
fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=a2) #1: 64, 2: 200
# The second hidden layer takes the
# result from act1 as input, with
# its name as "fc2" and the number
# of hidden neurons as 64.
act2 <- mx.symbol.Activation(fc2, name="relu2", act_type="relu")
# The second activation is almost
# the same as act1, except we
# have a different input source and name.
fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=a3) #1: 10, 2:100
# This generates the output layer.
# Because there are only 10
# digits, we set the number of neurons to 10.
act3 <- mx.symbol.Activation(fc3, name="relu3", act_type="relu")
# The third activation is almost the same as act3,
# except different layers.
fc4 <- mx.symbol.FullyConnected(act3, name="fc4", num_hidden=a4)
# This generates output layer.
softmax <- mx.symbol.SoftmaxOutput(fc4, name="sm")
# Finally, we set the activation
# to softmax to get a probabilistic prediction.

# Training:
# We are almost ready for the training process.
# Before we start the computation, let's decide which device to use:
devices <- mx.cpu()
# We assign CPU to mxnet.
# Now, you can run the following command
# to train the neural network!

# Note that mx.set.seed is the function
# that controls the random process in mxnet:

mx.set.seed(0)
model <- mx.model.FeedForward.create(
  softmax, X=train.x, y=train.y,
  ctx=devices, num.round=iter,
  array.batch.size=100,
  learning.rate=0.1, momentum=0.9,
  eval.metric=mx.metric.accuracy,
  initializer=mx.init.uniform(0.07),
  epoch.end.callback=mx.callback.log.train.metric(100)
  # epoch.end.callback=mx.callback.plot.train.metric(100, logger)
)
```

```
## Warning in mx.model.select.layout.train(X, y): Auto detect layout input matrix, use colmajor..
```

```
## Start training with 1 devices
## [1] Train-accuracy=0.4575
## [2] Train-accuracy=0.596
## [3] Train-accuracy=0.614
## [4] Train-accuracy=0.61
## [5] Train-accuracy=0.8
## [6] Train-accuracy=0.854
## [7] Train-accuracy=0.906
## [8] Train-accuracy=0.888
## [9] Train-accuracy=0.93
## [10] Train-accuracy=0.894
## [11] Train-accuracy=0.908
## [12] Train-accuracy=0.938
## [13] Train-accuracy=0.92
## [14] Train-accuracy=0.924
## [15] Train-accuracy=0.938
## [16] Train-accuracy=0.928
## [17] Train-accuracy=0.928
## [18] Train-accuracy=0.928
## [19] Train-accuracy=0.942
## [20] Train-accuracy=0.938
## [21] Train-accuracy=0.936
## [22] Train-accuracy=0.926
## [23] Train-accuracy=0.93
## [24] Train-accuracy=0.946
## [25] Train-accuracy=0.942
## [26] Train-accuracy=0.934
## [27] Train-accuracy=0.93
## [28] Train-accuracy=0.936
## [29] Train-accuracy=0.948
## [30] Train-accuracy=0.948
```

```
# Make prediction:
preds <- predict(model, test.x)
```

```
## Warning in mx.model.select.layout.predict(X, model): Auto detect layout input matrix, use colmajor..
```

```
# It is a matrix
# containing the desired classification
# probabilities from the output layer.
# To extract the maximum label for each row, use max.col:

pred.label <- max.col(t(preds)) - 1
table(pred.label, test.y)
```

```
##             test.y
## pred.label   0  1
##           0 83  7
##           1  4 19
```

```
percent <- sum(diag(table(pred.label, test.y)))/sum(table(pred.label, test.y))
percent; 1-percent
```

```
## [1] 0.9026549
```

```
## [1] 0.09734513
```

17.2 Convolutional Neural Network

```r
################# CNN #################################

# Load data:
all <- read.csv('F:/data_b_cancer/data.csv', header = TRUE)
all <- all[,-1] # Get rid of ID
colnames(all)[1] <- "Diagnosis"; head(all); dim(all); names(all)
```

```
##   Diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1         M       17.99        10.38         122.80    1001.0
## 2         M       20.57        17.77         132.90    1326.0
## 3         M       19.69        21.25         130.00    1203.0
## 4         M       11.42        20.38          77.58     386.1
## 5         M       20.29        14.34         135.10    1297.0
## 6         M       12.45        15.70          82.57     477.1
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1         0.11840          0.27760         0.3001             0.14710
## 2         0.08474          0.07864         0.0869             0.07017
## 3         0.10960          0.15990         0.1974             0.12790
## 4         0.14250          0.28390         0.2414             0.10520
## 5         0.10030          0.13280         0.1980             0.10430
## 6         0.12780          0.17000         0.1578             0.08089
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1        0.2419                0.07871    1.0950     0.9053        8.589
## 2        0.1812                0.05667    0.5435     0.7339        3.398
## 3        0.2069                0.05999    0.7456     0.7869        4.585
## 4        0.2597                0.09744    0.4956     1.1560        3.445
## 5        0.1809                0.05883    0.7572     0.7813        5.438
## 6        0.2087                0.07613    0.3345     0.8902        2.217
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1  153.40      0.006399        0.04904      0.05373           0.01587
## 2   74.08      0.005225        0.01308      0.01860           0.01340
## 3   94.03      0.006150        0.04006      0.03832           0.02058
## 4   27.23      0.009110        0.07458      0.05661           0.01867
## 5   94.44      0.011490        0.02461      0.05688           0.01885
## 6   27.19      0.007510        0.03345      0.03672           0.01137
##   symmetry_se fractal_dimension_se radius_worst texture_worst
## 1     0.03003             0.006193        25.38         17.33
## 2     0.01389             0.003532        24.99         23.41
## 3     0.02250             0.004571        23.57         25.53
## 4     0.05963             0.009208        14.91         26.50
## 5     0.01756             0.005115        22.54         16.67
## 6     0.02165             0.005082        15.47         23.75
##   perimeter_worst area_worst smoothness_worst compactness_worst
## 1          184.60     2019.0           0.1622            0.6656
## 2          158.80     1956.0           0.1238            0.1866
## 3          152.50     1709.0           0.1444            0.4245
## 4           98.87      567.7           0.2098            0.8663
## 5          152.20     1575.0           0.1374            0.2050
## 6          103.40      741.6           0.1791            0.5249
##   concavity_worst concave.points_worst symmetry_worst
## 1          0.7119               0.2654         0.4601
## 2          0.2416               0.1860         0.2750
## 3          0.4504               0.2430         0.3613
## 4          0.6869               0.2575         0.6638
## 5          0.4000               0.1625         0.2364
## 6          0.5355               0.1741         0.3985
##   fractal_dimension_worst
## 1                 0.11890
## 2                 0.08902
## 3                 0.08758
## 4                 0.17300
## 5                 0.07678
## 6                 0.12440
```

```
## [1] 569  31
```

```
##  [1] "Diagnosis"               "radius_mean"
##  [3] "texture_mean"            "perimeter_mean"
##  [5] "area_mean"               "smoothness_mean"
##  [7] "compactness_mean"        "concavity_mean"
##  [9] "concave.points_mean"     "symmetry_mean"
## [11] "fractal_dimension_mean"  "radius_se"
## [13] "texture_se"              "perimeter_se"
## [15] "area_se"                 "smoothness_se"
## [17] "compactness_se"          "concavity_se"
## [19] "concave.points_se"       "symmetry_se"
## [21] "fractal_dimension_se"    "radius_worst"
## [23] "texture_worst"           "perimeter_worst"
## [25] "area_worst"              "smoothness_worst"
## [27] "compactness_worst"       "concavity_worst"
## [29] "concave.points_worst"    "symmetry_worst"
## [31] "fractal_dimension_worst"
```

```
# Create Dummies:
all$Diagnosis <- ifelse(all$Diagnosis == "M", 1, 0)
head(all); dim(all); names(all) # Check!
```

```
##   Diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1         1       17.99        10.38         122.80    1001.0
## 2         1       20.57        17.77         132.90    1326.0
## 3         1       19.69        21.25         130.00    1203.0
## 4         1       11.42        20.38          77.58     386.1
## 5         1       20.29        14.34         135.10    1297.0
## 6         1       12.45        15.70          82.57     477.1
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1         0.11840          0.27760         0.3001             0.14710
## 2         0.08474          0.07864         0.0869             0.07017
## 3         0.10960          0.15990         0.1974             0.12790
## 4         0.14250          0.28390         0.2414             0.10520
## 5         0.10030          0.13280         0.1980             0.10430
## 6         0.12780          0.17000         0.1578             0.08089
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1        0.2419                0.07871    1.0950     0.9053        8.589
## 2        0.1812                0.05667    0.5435     0.7339        3.398
## 3        0.2069                0.05999    0.7456     0.7869        4.585
## 4        0.2597                0.09744    0.4956     1.1560        3.445
## 5        0.1809                0.05883    0.7572     0.7813        5.438
## 6        0.2087                0.07613    0.3345     0.8902        2.217
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1  153.40      0.006399        0.04904      0.05373           0.01587
## 2   74.08      0.005225        0.01308      0.01860           0.01340
## 3   94.03      0.006150        0.04006      0.03832           0.02058
## 4   27.23      0.009110        0.07458      0.05661           0.01867
## 5   94.44      0.011490        0.02461      0.05688           0.01885
## 6   27.19      0.007510        0.03345      0.03672           0.01137
##   symmetry_se fractal_dimension_se radius_worst texture_worst
## 1     0.03003             0.006193        25.38         17.33
## 2     0.01389             0.003532        24.99         23.41
## 3     0.02250             0.004571        23.57         25.53
## 4     0.05963             0.009208        14.91         26.50
## 5     0.01756             0.005115        22.54         16.67
## 6     0.02165             0.005082        15.47         23.75
##   perimeter_worst area_worst smoothness_worst compactness_worst
## 1          184.60     2019.0           0.1622            0.6656
## 2          158.80     1956.0           0.1238            0.1866
## 3          152.50     1709.0           0.1444            0.4245
## 4           98.87      567.7           0.2098            0.8663
## 5          152.20     1575.0           0.1374            0.2050
## 6          103.40      741.6           0.1791            0.5249
##   concavity_worst concave.points_worst symmetry_worst
## 1          0.7119               0.2654         0.4601
## 2          0.2416               0.1860         0.2750
## 3          0.4504               0.2430         0.3613
## 4          0.6869               0.2575         0.6638
## 5          0.4000               0.1625         0.2364
## 6          0.5355               0.1741         0.3985
##   fractal_dimension_worst
## 1                 0.11890
## 2                 0.08902
## 3                 0.08758
## 4                 0.17300
## 5                 0.07678
## 6                 0.12440
```

```
## [1] 569  31
```

```
##  [1] "Diagnosis"               "radius_mean"
##  [3] "texture_mean"            "perimeter_mean"
##  [5] "area_mean"               "smoothness_mean"
##  [7] "compactness_mean"        "concavity_mean"
##  [9] "concave.points_mean"     "symmetry_mean"
## [11] "fractal_dimension_mean"  "radius_se"
## [13] "texture_se"              "perimeter_se"
## [15] "area_se"                 "smoothness_se"
## [17] "compactness_se"          "concavity_se"
## [19] "concave.points_se"       "symmetry_se"
## [21] "fractal_dimension_se"    "radius_worst"
## [23] "texture_worst"           "perimeter_worst"
## [25] "area_worst"              "smoothness_worst"
## [27] "compactness_worst"       "concavity_worst"
## [29] "concave.points_worst"    "symmetry_worst"
## [31] "fractal_dimension_worst"
```

```r
# Set up:
all <- data.frame(
  all,
  matrix(0,nrow=nrow(all),ncol=((round(sqrt(ncol(all))))^2 - ncol(all) +1))
); dim(all)
```

```
## [1] 569  37
```

```r
# Load train and test datasets
train <- all[1:(0.8*nrow(all)),]; dim(train) # Training set
```

```
## [1] 455  37
```

```r
test <- all[(0.8*nrow(all)+1):nrow(all),]; dim(test) # Testing set
```

```
## [1] 113  37
```

```r
# Set up train and test datasets
train <- data.matrix(train)
train_x <- t(train[, -1])
train_y <- train[, 1]
train_array <- train_x
size <- round(sqrt(nrow(train_x)))
dim(train_array) <- c(size, size, 1, ncol(train_x))

test_x <- t(test[, -1])
test_y <- test[, 1]
test_array <- test_x
dim(test_array) <- c(size, size, 1, ncol(test_x))

# Set up the symbolic model
data <- mx.symbol.Variable('data')
conv_1 <- mx.symbol.Convolution(data = data, kernel = c(3,3), num_filter = 200)
tanh_1 <- mx.symbol.Activation(data = conv_1, act_type = "tanh")
pool_1 <- mx.symbol.Pooling(data = tanh_1, pool_type = "max", kernel = c(2,2), stride = c(2, 2))
# 2nd convolutional layer
conv_2 <- mx.symbol.Convolution(data = pool_1, kernel = c(1,1), num_filter = 100)
tanh_2 <- mx.symbol.Activation(data = conv_2, act_type = "tanh")
pool_2 <- mx.symbol.Pooling(data=tanh_2, pool_type = "max", kernel = c(2, 2), stride = c(2, 2))
# 1st fully connected layer
flatten <- mx.symbol.Flatten(data = pool_2)
fc_1 <- mx.symbol.FullyConnected(data = flatten, num_hidden = 100) # LeCun: 500
tanh_3 <- mx.symbol.Activation(data = fc_1, act_type = "tanh")
# 2nd fully connected layer
fc_2 <- mx.symbol.FullyConnected(data = tanh_3, num_hidden = 100)
# Output. Softmax output since we'd like to get some probabilities.
NN_model <- mx.symbol.SoftmaxOutput(data = fc_2)

# Pre-training set up:
# Set seed for reproducibility
mx.set.seed(100)

# Device used. CPU in my case.
devices <- mx.cpu()

# Training
```

```
iter <- 20

# Train the model
model <- mx.model.FeedForward.create(
  NN_model,
  X = train_array,
  y = train_y,
  ctx = devices,
  num.round = iter, # LeCun 480
  array.batch.size = 40,
  learning.rate = 0.01,
  momentum = 0.9,
  eval.metric = mx.metric.accuracy,
  epoch.end.callback = mx.callback.log.train.metric(100)
)
```

```
## Start training with 1 devices
## [1] Train-accuracy=0.538636363636364
## [2] Train-accuracy=0.585416666666667
## [3] Train-accuracy=0.585416666666667
## [4] Train-accuracy=0.585416666666667
## [5] Train-accuracy=0.585416666666667
## [6] Train-accuracy=0.53125
## [7] Train-accuracy=0.49375
## [8] Train-accuracy=0.49375
## [9] Train-accuracy=0.497916666666667
## [10] Train-accuracy=0.497916666666667
## [11] Train-accuracy=0.51875
## [12] Train-accuracy=0.51875
## [13] Train-accuracy=0.51875
## [14] Train-accuracy=0.51875
## [15] Train-accuracy=0.51875
## [16] Train-accuracy=0.51875
## [17] Train-accuracy=0.520833333333333
## [18] Train-accuracy=0.579166666666667
## [19] Train-accuracy=0.63125
## [20] Train-accuracy=0.664583333333333
```

```
# Testing:
# Predict labels
predicted <- predict(model, test_array)
# Assign labels
predicted_labels <- max.col(t(predicted)) - 1

table(predicted_labels, test_y)
```

```
##                   test_y
## predicted_labels  0  1
##                 0 87 16
##                 1  0 10
```

```
percent <- sum(diag(table(predicted_labels, test_y)))/sum(table(predicted_labels, test_y))
percent; 1-percent
```

```
## [1] 0.8584071
## [1] 0.1415929
```

18 Homework 1

18.1 Problem 1

A classifier, say f, is a mapping from a feature space $\mathbb{X} = \mathbb{R}^d$ to label space $\mathcal{Y}$. The loss of this classifier using 0-1 loss is defined as the following:

$$L(\hat{y}, y) = 1\{\hat{y} \neq y\} = P_{XY}(f(X) \neq Y).$$

The risk, the expected value of the loss function, is defined as

$$R(f) = E[L(f(X), Y)] = E[1_{\{f(X) \neq Y\}}] = P(f(X) \neq Y)$$

Given from lecture, we define Bayes' Classifier to be the following mapping:

$$f^*(x) = \begin{cases} 1, & \eta(x) \geq 1/2 \\ 0, & \text{otherwise} \end{cases}$$

where $\eta(x) \equiv P_{Y|X}(Y = 1|X = x)$.The goal is to prove the statement: Bayes classifier is the optimal classifier than any other classifier.

Proof:

Let $f : \mathcal{X} \to \mathcal{Y}$ be any classifier. We want to show that $R(f) - R(f^*) \geq 0$, i.e. the Bayes classifier performs better than any other classifiers.

Following definition, we have

$$\begin{aligned} R(f) - R(f^*) &= P(f(X) \neq Y) - P(f^*(X) \neq Y) \\ &= \int (P(f(X) \neq Y|X = x) - P(f^*(X) \neq Y|X = x))p(x)dx \end{aligned}$$

and it is sufficient to prove the argument by proving $P(f(X) \neq Y|X = x) - P(f^*(X) \neq Y|X = x) \geq 0$. Starting with the following, for any f, we have

$$\begin{aligned} P(f(X) \neq Y|X = x) &= 1 - P(f(X) = Y|X = x) \\ &= 1 - [P(Y = 1, f(X) = 1|X = x) + P(Y = 0, f(X) = 0|X = x)] \\ &= 1 - [\mathbb{E}(\mathbf{1}\{Y = 1\}\mathbf{1}\{f(X) = 1\}|X = x) + \mathbb{E}(\mathbf{1}\{Y = 0\}\mathbf{1}\{f(X) = 0\}|X = x)] \\ &= 1 - [\mathbf{1}\{f(X) = 1\}P(Y = 1|X = x) + \mathbf{1}\{f(X) = 0\}P(Y = 0|X = x)] \\ &= 1 - [\mathbf{1}\{f(X) = 1\}\eta(x) + \mathbf{1}\{f(X) = 0\}(1 - \eta(x))], \forall f \end{aligned}$$

and we know that for Bayes, $f^*(X)$ scenario takes the same format. Hence, we have the following

$$\begin{aligned} R(f) - R(f^*) &= \int (P(f(X) \neq Y|X = x) - P(f^*(X) \neq Y|X = x))p(x)dx \\ &= 1 - [\mathbf{1}\{f(X) = 1\}\eta(x) + \mathbf{1}\{f(X) = 0\}(1 - \eta(x))] \\ &\quad -\{1 - [\mathbf{1}\{f^*(X) = 1\}\eta(x) + \mathbf{1}\{f^*(X) = 0\}(1 - \eta(x))]\} \\ &= \eta(x)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] + (1 - \eta(x))[\mathbf{1}\{f^*(X) = 0\} - \mathbf{1}\{f(X) = 0\}] \\ &= \eta(x)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] + (\eta(x) - 1)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] \\ &= (\eta(x) + \eta(x) - 1)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] \\ &= (2\eta(x) - 1)[\mathbf{1}\{f^*(X) = 1\} - \mathbf{1}\{f(X) = 1\}] \end{aligned}$$

and we discuss the following cases

$$\begin{cases} (2\eta(x)-1)[\mathbf{1}\{f^*(X)=1\}-\mathbf{1}\{f(X)=1\}]\geq 0, & \text{if } \eta(x)\geq 1/2 \text{ since all components are greater or equal to zero} \\ (2\eta(x)-1)[\mathbf{1}\{f^*(X)=1\}-\mathbf{1}\{f(X)=1\}]\geq 0, & \text{if } \eta(x)< 1/2 \text{ since all components are less or equal to zero} \end{cases}$$

This, therefore, implies that $R(f) - R(f^*) \geq 0$.

□

18.2 Problem 2

18.2.1 1. Download Data

We want to download 30 stocks of DJIA with closing prices for every trading day from Jan. 1 2010 to Jan. 1, 2011.

```
# Use Quantmod to download data:
# install.packages('quantmod')
require('quantmod')
```

```
## Loading required package: quantmod

## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: TTR

## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
# Download and save all 30 companies in a vector called "data":
# "data" simply stores the name of 30 companies.
data<-getSymbols(
    c(
      "AAPL", "AXP", "BA", "CAT", "CSCO",
      "CVX", "DD", "DIS", "GE", "GS",
      "HD", "IBM", "INTC", "JNJ", "JPM",
      "KO", "MCD", "MMM", "MRK", "MSFT",
      "NKE", "PFE", "PG", "TRV", "UNH",
      "UTX", "V", "VZ", "WMT", "XOM"
    ),
    src="google",
    from=as.Date("2010-01-01"),
    to=as.Date("2011-04-04")
); length(data)
```

```
##     As of 0.4-0, 'getSymbols' uses env=parent.frame() and
##  auto.assign=TRUE by default.
##
##  This  behavior  will be  phased out in 0.5-0  when the call  will
##  default to use auto.assign=FALSE. getOption("getSymbols.env") and
##  getOptions("getSymbols.auto.assign") are now checked for alternate defaults
##
##  This message is shown once per session and may be disabled by setting
##  options("getSymbols.warning4.0"=FALSE). See ?getSymbols for more details.
```

```
## [1] 30
```

```
# As an example:
head(AAPL) # AAPL is a vector that gives five different information about this company.
```

```
##            AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume
## 2010-01-04     30.49     30.64    30.34      30.57   123432050
## 2010-01-05     30.66     30.80    30.46      30.63   150476004
## 2010-01-06     30.63     30.75    30.11      30.14   138039594
## 2010-01-07     30.25     30.29    29.86      30.08   119282324
## 2010-01-08     30.04     30.29    29.87      30.28   111969081
## 2010-01-11     30.40     30.43    29.78      30.02   115557365
```

```
# For example, say AAPL:
head(AAPL[,4]) # Check that the 4th column is closing price
```

```
##            AAPL.Close
## 2010-01-04      30.57
## 2010-01-05      30.63
## 2010-01-06      30.14
## 2010-01-07      30.08
## 2010-01-08      30.28
## 2010-01-11      30.02
```

```
data <- data.frame(data); dim(data)
```

```
## [1] 30  1
```

```
closing_mat <- data.frame(
  cbind(
    AAPL[,4], AXP[,4], BA[,4], CAT[,4], CSCO[,4],
    CVX[,4], DD[,4], DIS[,4], GE[,4], GS[,4],
    HD[,4], IBM[,4], INTC[,4], JNJ[,4], JPM[,4],
    KO[,4], MCD[,4], MMM[,4], MRK[,4], MSFT[,4],
    NKE[,4], PFE[,4], PG[,4], TRV[,4], UNH[,4],
    UTX[,4], V[,4], VZ[,4], WMT[,4], XOM[,4]
  )
); dim(closing_mat) # This is a matrix of all closing price for 30 companies.
```

```
## [1] 316  30
```

18.2.2 2. PCA on Prices (cor = “”)

We perform PCA on prices and create biplot

```
# PCA:
# ?princomp # To read and to understand the function
head(closing_mat); dim(closing_mat)
```

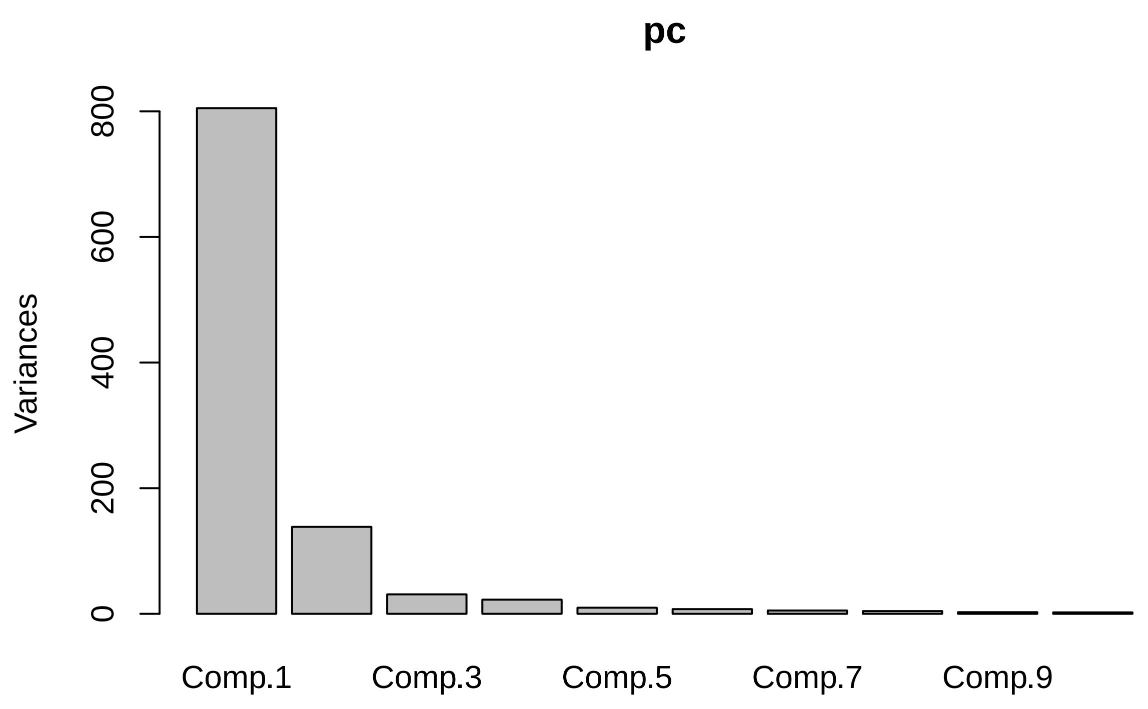
```
##            AAPL.Close AXP.Close BA.Close CAT.Close CSCO.Close CVX.Close
## 2010-01-04      30.57     40.92    56.18     58.55      24.69     79.06
## 2010-01-05      30.63     40.83    58.02     59.25      24.58     79.62
## 2010-01-06      30.14     41.49    59.78     59.43      24.42     79.63
## 2010-01-07      30.08     41.98    62.20     59.67      24.53     79.33
## 2010-01-08      30.28     41.95    61.60     60.34      24.66     79.47
## 2010-01-11      30.02     41.47    60.87     64.13      24.59     80.88
##            DD.Close DIS.Close GE.Close GS.Close HD.Close IBM.Close
## 2010-01-04    34.26     32.07    15.45   173.08    28.67    132.45
## 2010-01-05    33.93     31.99    15.53   176.14    28.88    130.85
## 2010-01-06    34.04     31.82    15.45   174.26    28.78    130.00
## 2010-01-07    34.39     31.83    16.25   177.67    29.12    129.55
## 2010-01-08    33.94     31.88    16.60   174.31    28.98    130.85
## 2010-01-11    34.26     31.36    16.76   171.56    28.16    129.48
##            INTC.Close JNJ.Close JPM.Close KO.Close MCD.Close MMM.Close
## 2010-01-04      20.88     64.68     42.85    28.52     62.78     83.02
## 2010-01-05      20.87     63.93     43.68    28.18     62.30     82.50
## 2010-01-06      20.80     64.45     43.92    28.16     61.45     83.67
## 2010-01-07      20.60     63.99     44.79    28.10     61.90     83.73
## 2010-01-08      20.83     64.21     44.68    27.58     61.84     84.32
## 2010-01-11      20.95     64.22     44.53    28.14     62.32     83.98
##            MRK.Close MSFT.Close NKE.Close PFE.Close PG.Close TRV.Close
## 2010-01-04     37.01      30.95     16.34     18.93    61.12     49.81
## 2010-01-05     37.16      30.96     16.40     18.66    61.14     48.63
## 2010-01-06     37.66      30.77     16.30     18.60    60.85     47.94
## 2010-01-07     37.72      30.45     16.46     18.53    60.52     48.63
## 2010-01-08     37.70      30.66     16.43     18.68    60.44     48.56
## 2010-01-11     37.85      30.27     16.23     18.83    60.20     48.54
##            UNH.Close UTX.Close V.Close VZ.Close WMT.Close XOM.Close
## 2010-01-04     31.53     71.63   22.04    33.28     54.23     69.15
## 2010-01-05     31.48     70.56   21.78    33.34     53.69     69.42
## 2010-01-06     31.79     70.19   21.49    31.92     53.57     70.02
## 2010-01-07     33.01     70.49   21.69    31.73     53.60     69.80
## 2010-01-08     32.70     70.63   21.75    31.75     53.33     69.52
## 2010-01-11     32.92     72.16   21.69    31.88     54.21     70.30
```

```
## [1] 316  30
```

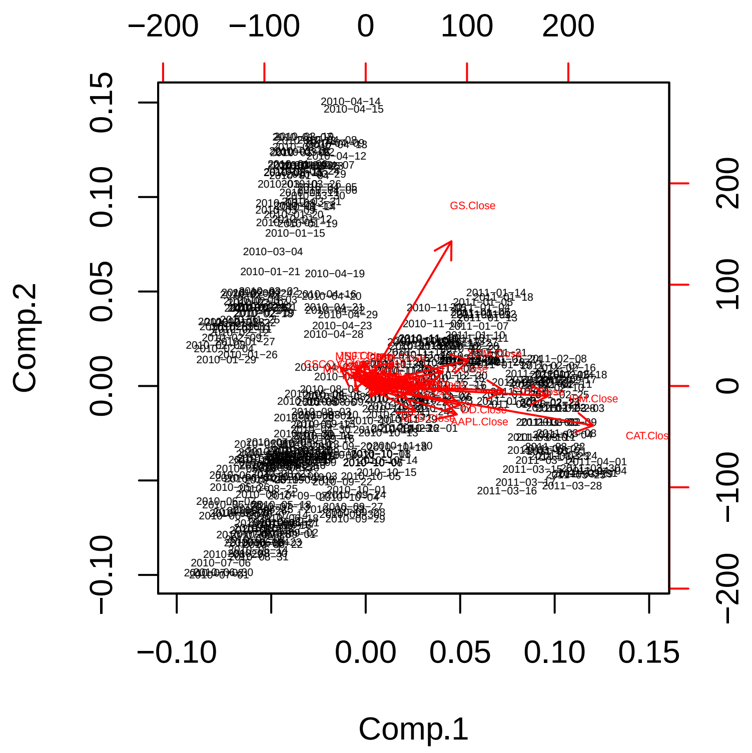
```
pc <- princomp(na.omit(closing_mat), cor=FALSE)
summary(pc)
```

```
## Importance of components:
##                           Comp.1     Comp.2     Comp.3     Comp.4
## Standard deviation     28.3724866 11.7654938 5.56807822 4.74678845
## Proportion of Variance  0.7756353  0.1333777 0.02987263 0.02171013
## Cumulative Proportion   0.7756353  0.9090129 0.93888557 0.96059571
##                             Comp.5      Comp.6      Comp.7      Comp.8
## Standard deviation     3.119792031 2.732214453 2.283856104 2.076552918
## Proportion of Variance 0.009378082 0.007192706 0.005025743 0.004154787
## Cumulative Proportion  0.969973788 0.977166495 0.982192237 0.986347024
##                             Comp.9    Comp.10     Comp.11     Comp.12
## Standard deviation     1.609704046 1.50371442 1.209215752 1.136287494
## Proportion of Variance 0.002496634 0.00217868 0.001408868 0.001244054
## Cumulative Proportion  0.988843658 0.99102234 0.992431206 0.993675260
##                           Comp.13     Comp.14     Comp.15      Comp.16
## Standard deviation     1.104842826 1.020725410 0.883062002 0.8058509989
## Proportion of Variance 0.001176153 0.001003877 0.000751355 0.0006257088
## Cumulative Proportion  0.994851413 0.995855290 0.996606645 0.9972323541
##                            Comp.17      Comp.18      Comp.19      Comp.20
## Standard deviation     0.7022425359 0.6645263232 0.6069514453 0.5292862973
## Proportion of Variance 0.0004751569 0.0004254878 0.0003549528 0.0002699256
## Cumulative Proportion  0.9977075110 0.9981329988 0.9984879517 0.9987578773
##                             Comp.21      Comp.22      Comp.23      Comp.24
## Standard deviation     0.5047136440 0.4761181275 0.4051941761 0.3832435809
## Proportion of Variance 0.0002454442 0.0002184199 0.0001581937 0.0001415183
## Cumulative Proportion  0.9990033215 0.9992217414 0.9993799351 0.9995214534
##                             Comp.25      Comp.26      Comp.27      Comp.28
## Standard deviation     0.3585348426 0.3249492043 3.005122e-01 2.732095e-01
## Proportion of Variance 0.0001238584 0.0001017405 8.701354e-05 7.192077e-05
## Cumulative Proportion  0.9996453118 0.9997470522 9.998341e-01 9.999060e-01
##                             Comp.29      Comp.30
## Standard deviation     2.545529e-01 1.810394e-01
## Proportion of Variance 6.243368e-05 3.157976e-05
## Cumulative Proportion  9.999684e-01 1.000000e+00
```

```
plot(pc)
```



```
biplot(pc, cex=.3)
```



```r
# Formula interface
princomp(~., data = closing_mat)
```

```
## Call:
## princomp(formula = ~., data = closing_mat)
##
## Standard deviations:
##     Comp.1     Comp.2     Comp.3     Comp.4     Comp.5     Comp.6
## 28.3724866 11.7654938  5.5680782  4.7467885  3.1197920  2.7322145
##     Comp.7     Comp.8     Comp.9    Comp.10    Comp.11    Comp.12
##  2.2838561  2.0765529  1.6097040  1.5037144  1.2092158  1.1362875
##    Comp.13    Comp.14    Comp.15    Comp.16    Comp.17    Comp.18
##  1.1048428  1.0207254  0.8830620  0.8058510  0.7022425  0.6645263
##    Comp.19    Comp.20    Comp.21    Comp.22    Comp.23    Comp.24
##  0.6069514  0.5292863  0.5047136  0.4761181  0.4051942  0.3832436
##    Comp.25    Comp.26    Comp.27    Comp.28    Comp.29    Comp.30
##  0.3585348  0.3249492  0.3005122  0.2732095  0.2545529  0.1810394
##
##  30  variables and  315 observations.
```

18.2.3 3. PCA on Prices (cor = TRUE)

We perform PCA on prices and create biplot

```r
# PCA:
# ?princomp # To read and to understand the function
```

```r
head(closing_mat); dim(closing_mat)
```

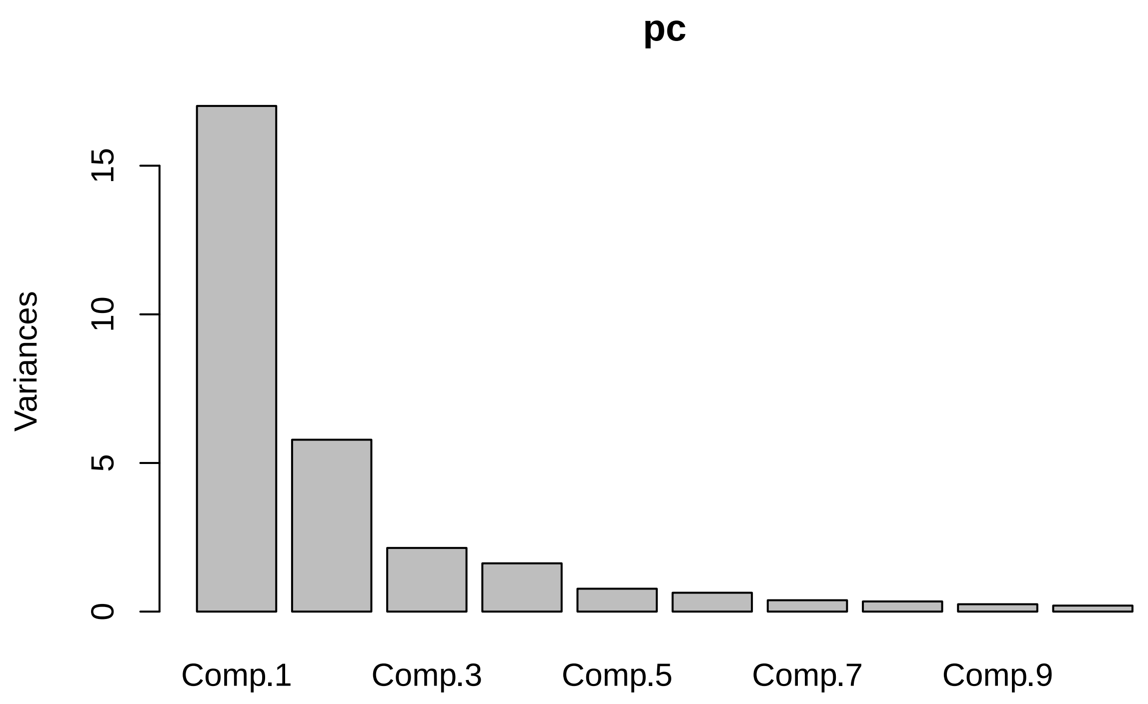
```
##            AAPL.Close AXP.Close BA.Close CAT.Close CSCO.Close CVX.Close
## 2010-01-04      30.57     40.92    56.18     58.55      24.69     79.06
## 2010-01-05      30.63     40.83    58.02     59.25      24.58     79.62
## 2010-01-06      30.14     41.49    59.78     59.43      24.42     79.63
## 2010-01-07      30.08     41.98    62.20     59.67      24.53     79.33
## 2010-01-08      30.28     41.95    61.60     60.34      24.66     79.47
## 2010-01-11      30.02     41.47    60.87     64.13      24.59     80.88
##            DD.Close DIS.Close GE.Close GS.Close HD.Close IBM.Close
## 2010-01-04    34.26     32.07    15.45   173.08    28.67    132.45
## 2010-01-05    33.93     31.99    15.53   176.14    28.88    130.85
## 2010-01-06    34.04     31.82    15.45   174.26    28.78    130.00
## 2010-01-07    34.39     31.83    16.25   177.67    29.12    129.55
## 2010-01-08    33.94     31.88    16.60   174.31    28.98    130.85
## 2010-01-11    34.26     31.36    16.76   171.56    28.16    129.48
##            INTC.Close JNJ.Close JPM.Close KO.Close MCD.Close MMM.Close
## 2010-01-04      20.88     64.68     42.85    28.52     62.78     83.02
## 2010-01-05      20.87     63.93     43.68    28.18     62.30     82.50
## 2010-01-06      20.80     64.45     43.92    28.16     61.45     83.67
## 2010-01-07      20.60     63.99     44.79    28.10     61.90     83.73
## 2010-01-08      20.83     64.21     44.68    27.58     61.84     84.32
## 2010-01-11      20.95     64.22     44.53    28.14     62.32     83.98
##            MRK.Close MSFT.Close NKE.Close PFE.Close PG.Close TRV.Close
## 2010-01-04     37.01      30.95     16.34     18.93    61.12     49.81
## 2010-01-05     37.16      30.96     16.40     18.66    61.14     48.63
## 2010-01-06     37.66      30.77     16.30     18.60    60.85     47.94
## 2010-01-07     37.72      30.45     16.46     18.53    60.52     48.63
## 2010-01-08     37.70      30.66     16.43     18.68    60.44     48.56
## 2010-01-11     37.85      30.27     16.23     18.83    60.20     48.54
##            UNH.Close UTX.Close V.Close VZ.Close WMT.Close XOM.Close
## 2010-01-04     31.53     71.63   22.04    33.28     54.23     69.15
## 2010-01-05     31.48     70.56   21.78    33.34     53.69     69.42
## 2010-01-06     31.79     70.19   21.49    31.92     53.57     70.02
## 2010-01-07     33.01     70.49   21.69    31.73     53.60     69.80
## 2010-01-08     32.70     70.63   21.75    31.75     53.33     69.52
## 2010-01-11     32.92     72.16   21.69    31.88     54.21     70.30
```

```
## [1] 316  30
```

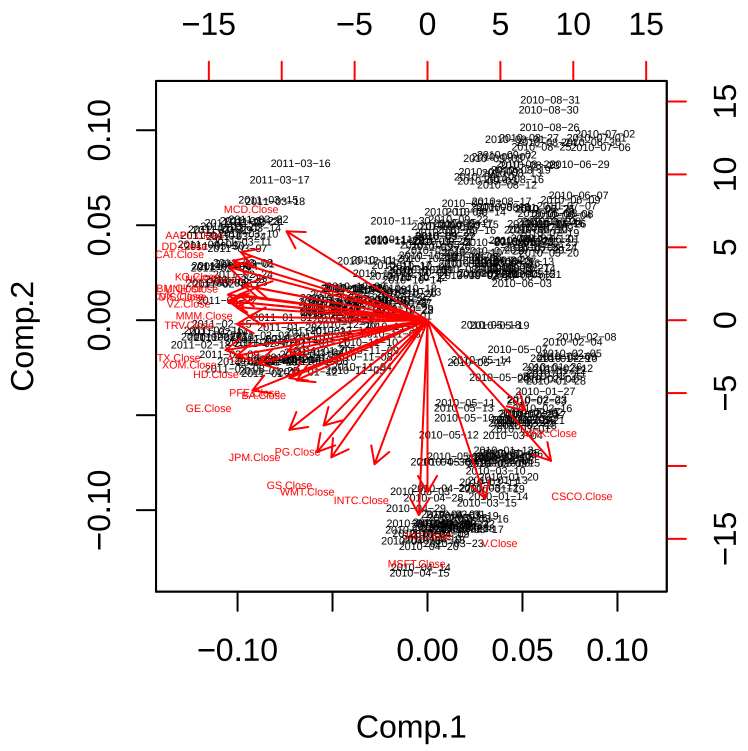
```r
pc <- princomp(na.omit(closing_mat), cor = TRUE)
summary(pc)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3    Comp.4     Comp.5
## Standard deviation     4.1243249 2.4043963 1.46324939 1.2742680 0.87725345
## Proportion of Variance 0.5670019 0.1927041 0.07136996 0.0541253 0.02565245
## Cumulative Proportion  0.5670019 0.7597059 0.83107589 0.8852012 0.91085365
##                            Comp.6     Comp.7     Comp.8      Comp.9
## Standard deviation     0.79625421 0.61814142 0.58452373 0.497162769
## Proportion of Variance 0.02113403 0.01273663 0.01138893 0.008239027
## Cumulative Proportion  0.93198767 0.94472430 0.95611323 0.964352259
##                            Comp.10     Comp.11    Comp.12     Comp.13
## Standard deviation     0.449065971 0.413757446 0.36776909 0.328794824
## Proportion of Variance 0.006722008 0.005706507 0.00450847 0.003603535
## Cumulative Proportion  0.971074267 0.976780775 0.98128925 0.984892780
##                            Comp.14     Comp.15     Comp.16     Comp.17
## Standard deviation     0.284580982 0.241605993 0.229765359 0.217367596
## Proportion of Variance 0.002699545 0.001945782 0.001759737 0.001574956
## Cumulative Proportion  0.987592324 0.989538106 0.991297843 0.992872799
##                            Comp.18     Comp.19      Comp.20      Comp.21
## Standard deviation     0.198765625 0.193123560 0.1603428985 0.1540532115
## Proportion of Variance 0.001316926 0.001243224 0.0008569948 0.0007910797
## Cumulative Proportion  0.994189725 0.995432948 0.9962899433 0.9970810230
##                             Comp.22      Comp.23      Comp.24      Comp.25
## Standard deviation     0.1295486920 0.1267251639 0.1152138867 0.1081783324
## Proportion of Variance 0.0005594288 0.0005353089 0.0004424747 0.0003900851
## Cumulative Proportion  0.9976404518 0.9981757607 0.9986182354 0.9990083204
##                             Comp.26     Comp.27      Comp.28      Comp.29
## Standard deviation     0.0970264279 0.089130465 0.0716860474 0.0658462908
## Proportion of Variance 0.0003138043 0.000264808 0.0001712963 0.0001445245
## Cumulative Proportion  0.9993221247 0.999586933 0.9997582290 0.9999027535
##                             Comp.30
## Standard deviation     5.401293e-02
## Proportion of Variance 9.724655e-05
## Cumulative Proportion  1.000000e+00
```

```r
plot(pc)
```



```r
biplot(pc, cex=.3)
```



```
# Comment: using cor=TRUE is setting the function to
# calculate principle component using correlation
# or covariance matrix. Observe the graph in the follow.
# It gives us better visualization of how the first
# and the second principle components affect V.

# Formula interface
princomp(~., data = closing_mat, cor = TRUE)
```

```
## Call:
## princomp(formula = ~., data = closing_mat, cor = TRUE)
##
## Standard deviations:
##     Comp.1     Comp.2     Comp.3     Comp.4     Comp.5     Comp.6
## 4.12432493 2.40439635 1.46324939 1.27426804 0.87725345 0.79625421
##     Comp.7     Comp.8     Comp.9    Comp.10    Comp.11    Comp.12
## 0.61814142 0.58452373 0.49716277 0.44906597 0.41375745 0.36776909
##    Comp.13    Comp.14    Comp.15    Comp.16    Comp.17    Comp.18
## 0.32879482 0.28458098 0.24160599 0.22976536 0.21736760 0.19876562
##    Comp.19    Comp.20    Comp.21    Comp.22    Comp.23    Comp.24
## 0.19312356 0.16034290 0.15405321 0.12954869 0.12672516 0.11521389
##    Comp.25    Comp.26    Comp.27    Comp.28    Comp.29    Comp.30
## 0.10817833 0.09702643 0.08913047 0.07168605 0.06584629 0.05401293
##
##  30  variables and  315 observations.
```

18.2.4 4. Return Analysis

Define return of a stock at a particular day, say day2, to be (closing price of day 2 - closing price of day 1)/closing price of day 1). Then we repeat part 4.

```r
# Compute return matrix
head(closing_mat,3); dim(closing_mat)
```

```
##            AAPL.Close AXP.Close BA.Close CAT.Close CSCO.Close CVX.Close
## 2010-01-04      30.57     40.92    56.18     58.55      24.69     79.06
## 2010-01-05      30.63     40.83    58.02     59.25      24.58     79.62
## 2010-01-06      30.14     41.49    59.78     59.43      24.42     79.63
##            DD.Close DIS.Close GE.Close GS.Close HD.Close IBM.Close
## 2010-01-04    34.26     32.07    15.45   173.08    28.67    132.45
## 2010-01-05    33.93     31.99    15.53   176.14    28.88    130.85
## 2010-01-06    34.04     31.82    15.45   174.26    28.78    130.00
##            INTC.Close JNJ.Close JPM.Close KO.Close MCD.Close MMM.Close
## 2010-01-04      20.88     64.68     42.85    28.52     62.78     83.02
## 2010-01-05      20.87     63.93     43.68    28.18     62.30     82.50
## 2010-01-06      20.80     64.45     43.92    28.16     61.45     83.67
##            MRK.Close MSFT.Close NKE.Close PFE.Close PG.Close TRV.Close
## 2010-01-04     37.01      30.95     16.34     18.93    61.12     49.81
## 2010-01-05     37.16      30.96     16.40     18.66    61.14     48.63
## 2010-01-06     37.66      30.77     16.30     18.60    60.85     47.94
##            UNH.Close UTX.Close V.Close VZ.Close WMT.Close XOM.Close
## 2010-01-04     31.53     71.63   22.04    33.28     54.23     69.15
## 2010-01-05     31.48     70.56   21.78    33.34     53.69     69.42
## 2010-01-06     31.79     70.19   21.49    31.92     53.57     70.02
```

```
## [1] 316  30
```

```r
return_mat <- matrix(0, nrow=315, ncol=ncol(closing_mat))
for (j in 1:ncol(closing_mat)){
  # Ex: j <- 1
  unit_return <- diff(closing_mat[,j])/lag(closing_mat[-1,][,j])
  return_mat[,j] <- unit_return
}; head(return_mat, 3); dim(return_mat)
```

```
##              [,1]         [,2]       [,3]        [,4]         [,5]
## [1,]  0.001958864 -0.002204262 0.03171320 0.011814346 -0.004475183
## [2,] -0.016257465  0.015907448 0.02944128 0.003028773 -0.006552007
## [3,] -0.001994681  0.011672225 0.03890675 0.004022122  0.004484305
##               [,6]         [,7]         [,8]         [,9]       [,10]
## [1,]  0.0070334087 -0.009725906 -0.002500781  0.005151320  0.01737254
## [2,]  0.0001255808  0.003231492 -0.005342552 -0.005177994 -0.01078848
## [3,] -0.0037816715  0.010177377  0.000314169  0.049230769  0.01919289
##             [,11]        [,12]         [,13]        [,14]       [,15]
## [1,]  0.007271468 -0.012227742 -0.0004791567 -0.011731581 0.019001832
## [2,] -0.003474635 -0.006538462 -0.0033653846  0.008068270 0.005464481
## [3,]  0.011675824 -0.003473562 -0.0097087379 -0.007188623 0.019423979
##              [,16]        [,17]        [,18]       [,19]         [,20]
## [1,] -0.0120652945 -0.007704655 -0.006303030 0.004036598  0.0003229974
## [2,] -0.0007102273 -0.013832384  0.013983507 0.013276686 -0.0061748456
## [3,] -0.0021352313  0.007269790  0.000716589 0.001590668 -0.0105090312
##             [,21]        [,22]         [,23]       [,24]        [,25]
## [1,]  0.003658537 -0.014469453  0.0003271181 -0.02426486 -0.001588310
## [2,] -0.006134969 -0.003225806 -0.0047658176 -0.01439299  0.009751494
## [3,]  0.009720535 -0.003777658 -0.0054527429  0.01418877  0.036958497
##              [,26]        [,27]        [,28]         [,29]        [,30]
## [1,] -0.015164399 -0.011937557  0.001799640 -0.0100577389  0.003889369
## [2,] -0.005271406 -0.013494649 -0.044486216 -0.0022400597  0.008568980
## [3,]  0.004255923  0.009220839 -0.005988024  0.0005597015 -0.003151862
```

```
## [1] 315  30
```

```
# PCA:
names(return_mat) <-    c(
      "AAPL", "AXP", "BA", "CAT", "CSCO",
      "CVX", "DD", "DIS", "GE", "GS",
      "HD", "IBM", "INTC", "JNJ", "JPM",
      "KO", "MCD", "MMM", "MRK", "MSFT",
      "NKE", "PFE", "PG", "TRV", "UNH",
      "UTX", "V", "VZ", "WMT", "XOM"
    )
head(return_mat, 3); dim(return_mat)
```

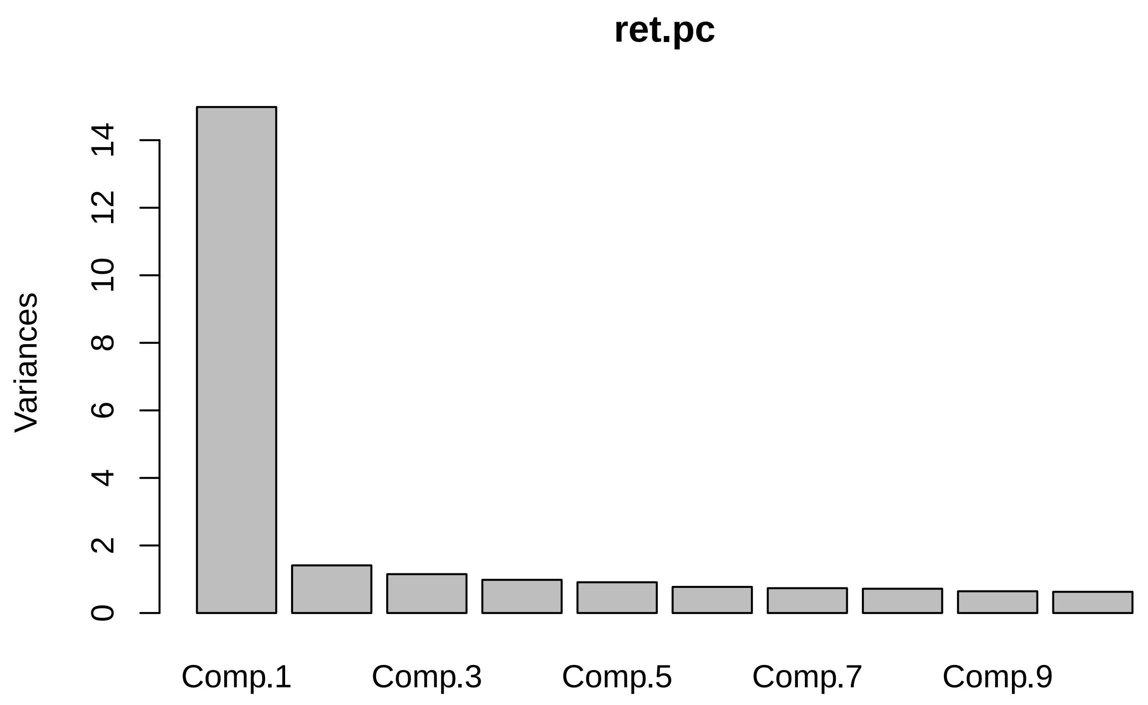
```
##              [,1]         [,2]       [,3]        [,4]         [,5]
## [1,]  0.001958864 -0.002204262 0.03171320 0.011814346 -0.004475183
## [2,] -0.016257465  0.015907448 0.02944128 0.003028773 -0.006552007
## [3,] -0.001994681  0.011672225 0.03890675 0.004022122  0.004484305
##               [,6]         [,7]         [,8]         [,9]       [,10]
## [1,]  0.0070334087 -0.009725906 -0.002500781  0.005151320  0.01737254
## [2,]  0.0001255808  0.003231492 -0.005342552 -0.005177994 -0.01078848
## [3,] -0.0037816715  0.010177377  0.000314169  0.049230769  0.01919289
##              [,11]        [,12]         [,13]        [,14]       [,15]
## [1,]  0.007271468 -0.012227742 -0.0004791567 -0.011731581 0.019001832
## [2,] -0.003474635 -0.006538462 -0.0033653846  0.008068270 0.005464481
## [3,]  0.011675824 -0.003473562 -0.0097087379 -0.007188623 0.019423979
##               [,16]        [,17]        [,18]       [,19]         [,20]
## [1,] -0.0120652945 -0.007704655 -0.006303030 0.004036598  0.0003229974
## [2,] -0.0007102273 -0.013832384  0.013983507 0.013276686 -0.0061748456
## [3,] -0.0021352313  0.007269790  0.000716589 0.001590668 -0.0105090312
##              [,21]        [,22]         [,23]       [,24]        [,25]
## [1,]  0.003658537 -0.014469453  0.0003271181 -0.02426486 -0.001588310
## [2,] -0.006134969 -0.003225806 -0.0047658176 -0.01439299  0.009751494
## [3,]  0.009720535 -0.003777658 -0.0054527429  0.01418877  0.036958497
##              [,26]        [,27]        [,28]         [,29]        [,30]
## [1,] -0.015164399 -0.011937557  0.001799640 -0.0100577389  0.003889369
## [2,] -0.005271406 -0.013494649 -0.044486216 -0.0022400597  0.008568980
## [3,]  0.004255923  0.009220839 -0.005988024  0.0005597015 -0.003151862
```

```
## [1] 315  30
```

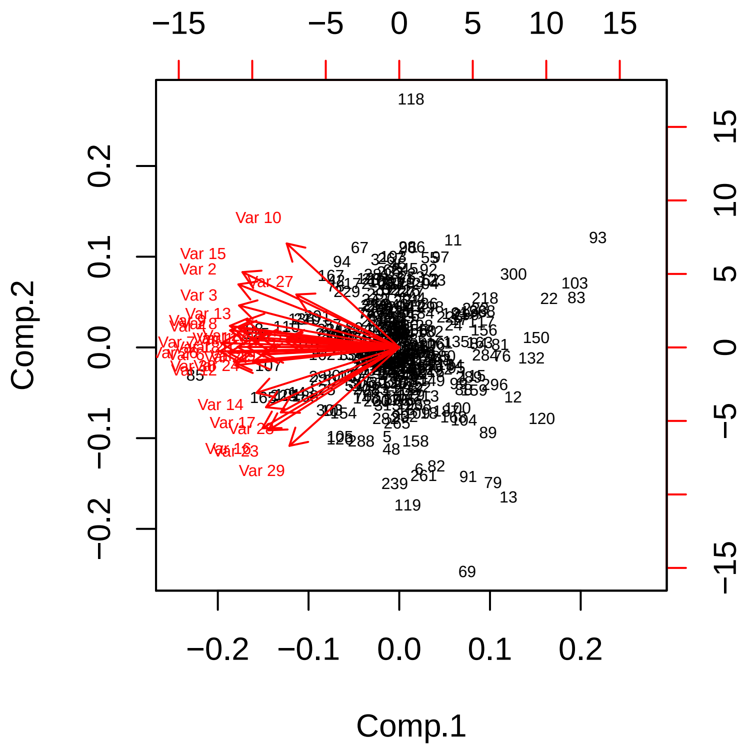
```
ret.pc <- princomp(na.omit(return_mat), cor = TRUE)
summary(ret.pc)
```

```
## Importance of components:
##                           Comp.1    Comp.2     Comp.3     Comp.4    Comp.5
## Standard deviation     3.8707234 1.1876754 1.07292952 0.99137358 0.9542861
## Proportion of Variance 0.4994166 0.0470191 0.03837259 0.03276072 0.0303554
## Cumulative Proportion  0.4994166 0.5464357 0.58480834 0.61756906 0.6479245
##                            Comp.6    Comp.7     Comp.8     Comp.9
## Standard deviation     0.87968107 0.8577325 0.84788132 0.80317836
## Proportion of Variance 0.02579463 0.0245235 0.02396342 0.02150318
## Cumulative Proportion  0.67371908 0.6982426 0.72220600 0.74370919
##                           Comp.10    Comp.11    Comp.12    Comp.13
## Standard deviation     0.79274340 0.76413727 0.73537579 0.70397704
## Proportion of Variance 0.02094807 0.01946353 0.01802592 0.01651946
## Cumulative Proportion  0.76465726 0.78412078 0.80214670 0.81866616
##                           Comp.14    Comp.15   Comp.16    Comp.17
## Standard deviation     0.69652957 0.69136189 0.6702859 0.65200137
## Proportion of Variance 0.01617178 0.01593271 0.0149761 0.01417019
## Cumulative Proportion  0.83483794 0.85077065 0.8657468 0.87991694
##                           Comp.18    Comp.19    Comp.20    Comp.21
## Standard deviation     0.62401472 0.62224893 0.60150949 0.57860049
## Proportion of Variance 0.01297981 0.01290646 0.01206046 0.01115928
## Cumulative Proportion  0.89289676 0.90580321 0.91786367 0.92902295
##                           Comp.22     Comp.23     Comp.24     Comp.25
## Standard deviation     0.56775069 0.542324649 0.537944584 0.514064709
## Proportion of Variance 0.01074469 0.009803868 0.009646146 0.008808751
## Cumulative Proportion  0.93976765 0.949571516 0.959217662 0.968026412
##                            Comp.26     Comp.27     Comp.28     Comp.29
## Standard deviation     0.502507035 0.465574070 0.444256418 0.407924386
## Proportion of Variance 0.008417111 0.007225307 0.006578792 0.005546743
## Cumulative Proportion  0.976443523 0.983668830 0.990247622 0.995794366
##                            Comp.30
## Standard deviation     0.355202789
## Proportion of Variance 0.004205634
## Cumulative Proportion  1.000000000
```

```r
plot(ret.pc)
```



```r
biplot(ret.pc, cex=.5)
```



```
# Comment: Now all 30 vectors are skewed towards (or close)
# to x-axis a lot more than the previous graph shown.
```

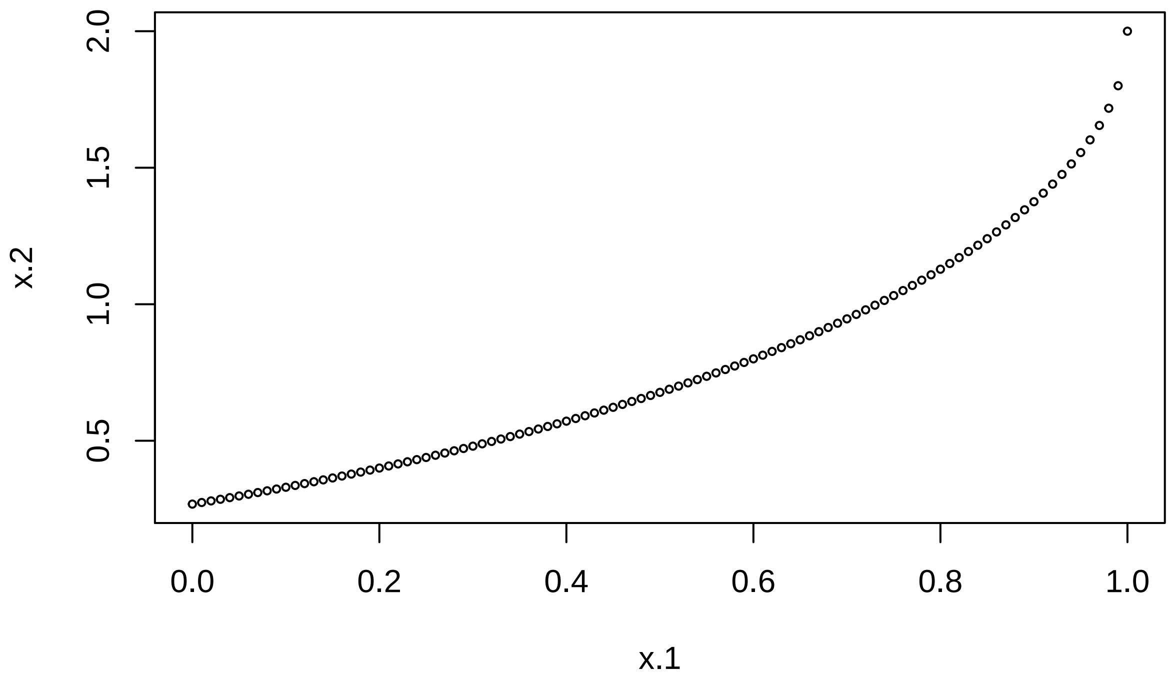
19 Homework 2

19.1 Problem 1

(a) Sketch Curve $(1+X_1)^2 + (2-X_2)^2 = 4$. We simplify the curve

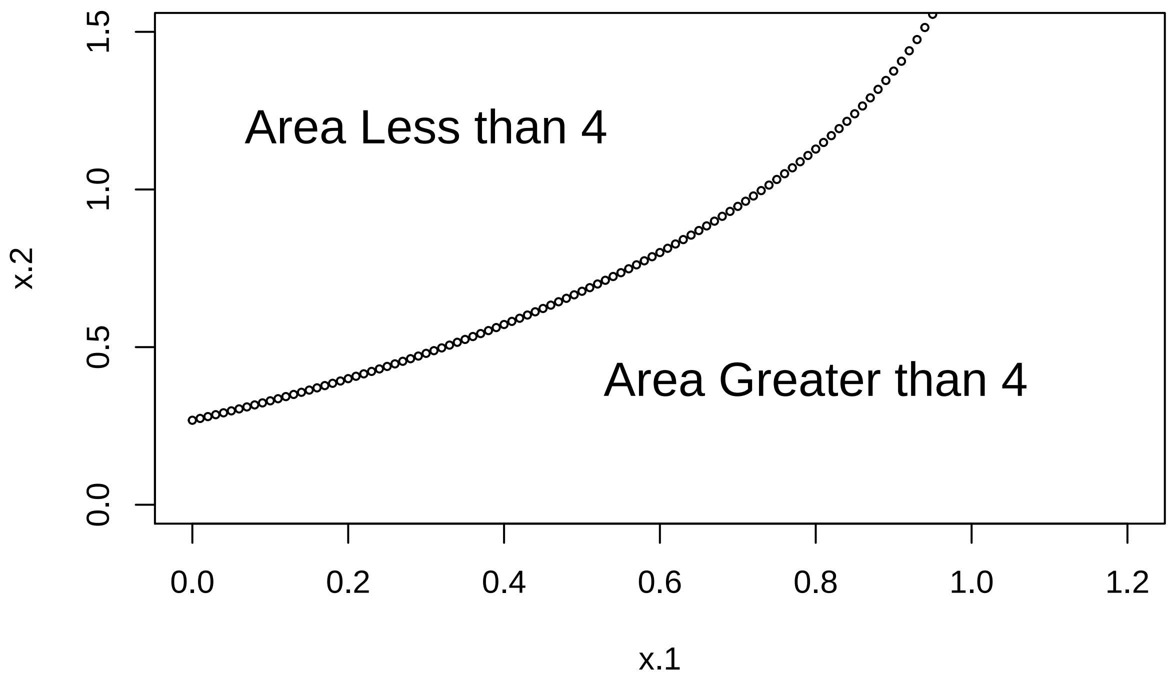
$$\begin{aligned}(1+X_1)^2 + (2-X_2)^2 &= 4\\ (2-X_2)^2 &= 4-(1+X_1)^2\\ 2-X_2 &= \sqrt{4-(1+X_1)^2}\\ X_2 &= 2-\sqrt{4-(1+X_1)^2}\end{aligned}$$

```r
# Plot the curve based on the above equation,
# Treat x.2 as a function of x.1:
x.1 <- seq(0,1,0.01)
x.2 <- 2 - (4 - (1+x.1)^2)^(1/2)
plot(x.1, x.2, cex=.5)
```



(b) Indicate

```r
x.1 <- seq(0,1,0.01)
x.2 <- 2 - (4 - (1+x.1)^2)^(1/2)
plot(x.1, x.2, cex=.5, xlim = c(0,1.2), ylim = c(0,1.5))
# points(0.4, 1.2, pch = "*", col = "purple")
text(0.3, 1.2, "Area Less than 4", cex = 1.5)
text(0.8, 0.4, "Area Greater than 4", cex = 1.5)
```



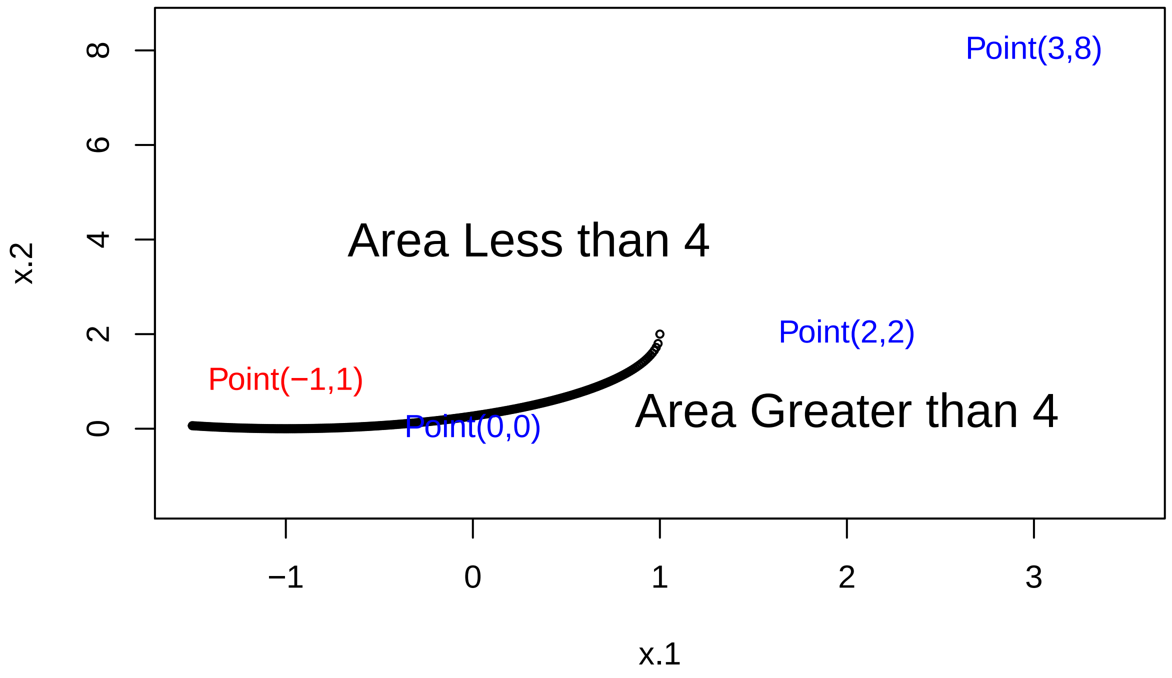
```
# Comment:
# The curve, designed in the problem, is a classifier
# to differentiate the value of the equation to be
# less than or greater than 4.
```

(c) Suppose a classifier assigns an observation to a blue class

$$(1 + X_1)^2 + (2 - X_2)^2 > 4$$

and to the red class otherwise. To what class are the observations $(0, 0)$, $(-1, -1)$, $(2, 2)$, or $(3, 8)$ classified?

```
# From (b), we have:
x.1 <- seq(-1.5,1.5,0.01)
x.2 <- 2 - (4 - (1+x.1)^2)^(1/2)
plot(x.1, x.2, cex=.5, xlim = c(-1.5,3.5), ylim = c(-1.5,8.5))
# points(0.4, 1.2, pch = "*", col = "purple")
text(0.3, 4, "Area Less than 4", cex = 1.5)
text(2, 0.4, "Area Greater than 4", cex = 1.5)
# Comment:
# The curve, designed in the problem, is a classifier
# to differentiate the value of the equation to be
# less than or greater than 4.

# For this problem, part (c), we simply add these
# points back to the graph:
text(0,0,"Point(0,0)",pch="*",col="Blue")
text(-1,1,"Point(-1,1)",pch="*",col="Red")
text(2,2,"Point(2,2)",pch="*",col="Blue")
text(3,8,"Point(3,8)",pch="*",col="Blue")
```



```
# Check math for sure:
1+0+2^2 > 4
```

```
## [1] TRUE
```

```
(1-1)^2+(2-1)^2 > 4
```

```
## [1] FALSE
```

```
(1+2)^2+(2-2)^2 > 4
```

```
## [1] TRUE
```

```
(1+3)^2 + (2-8)^2 > 4
```

```
## [1] TRUE
```

```
# Comment:
# To visualize this problem, we lie the points
# on the plot and we observe that all four points
# lie in the area that is greater than 4.
```

(d) Arguement for Linear vs Non-linear Decision Boundary

19.2 Problem 2

In this problem, we apply SVM to classify hand-written digits. We use R package “e1071” for SVM coding.

```r
# Load Data set:
require('e1071')
setwd("F://Course/CU Stats/STATS W4241(S) - Statistical Machine Learning/6. Homework/HW2")
train.5 <- as.matrix(read.table("train.5.txt", header = F, sep = ",")); dim(train.5)
```

```
## [1] 556 256
```

```r
train.6 <- as.matrix((read.table("train.6.txt", header = F, sep = ","))); dim(train.6)
```

```
## [1] 664 256
```

```r
# Read dimensions:
# 556x256 for digit 5
# 664x256 for digit 6

# Label "5" as -1, and "6" as 1:
explanatory <- rbind(
  train.5,
  train.6
); dim(explanatory)
```

```
## [1] 1220  256
```

```r
response <- rbind(
  matrix(-1,nrow=556,ncol=1),
  matrix(1,nrow=664,ncol=1)
); dim(response)
```

```
## [1] 1220    1
```

```r
# Create dataset:
data <- cbind(response,explanatory); dim(data)
```

```
## [1] 1220  257
```

```r
data <- data[sample(nrow(data), nrow(data)), ]; dim(data)
```

```
## [1] 1220  257
```

```r
# Set 80% training 20% testing:
train <- data[1:(.8*nrow(data)),]
test <- data[(.8*nrow(data)+1):nrow(data),]
```

```r
# SVM:
# I wrote a function called manual.tune, that is,
# it is a function allowing me to enter different
# parameters: cost = c, gamma = g:
manual.tune <- function(c,g){
  ## Apply SVM
  # Ex: c<-1; g<-1
  svm.fit <- svm(
    formula = train[,1] ~.,
    data = train[,-1],
    type = "C-classification",
    kernel = "linear",
    cost=c,
    gamma=g
    )

  ## Now we predict by the model above:
  pred <- predict(
    svm.fit,
    newdata = data.frame(test)
  )

  ## Visualize:
  table <- table(predict=pred, truth=test[,1])
  # roc <- plot.roc(pred, data.test$DRIVER, col="green")

  ## Compute coverage:
  cover.percentile <- sum(diag(table))/sum(table)

  ## Return:
  return(list(
    Summary = summary(svm.fit),
    Table = table,
    Accuracy = cover.percentile))
  }
## End of function

# Ex:
manual.tune(2,1)
```

```
## $Summary
##
## Call:
## svm(formula = train[, 1] ~ ., data = train[, -1], type = "C-classification",
##     kernel = "linear", cost = c, gamma = g)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  2
##       gamma:  1
##
## Number of Support Vectors:  82
##
##  ( 39 43 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
##
##
##
##
```

```
## $Table
##        truth
## predict  -1   1
##      -1 107 133
##      1    4   0
##
## $Accuracy
## [1] 0.4385246
```

19.2.1 (a) Cross-Validation (Linear)

Here we test Linear case:

```
# Parameters:
start <- 2; end <- 4; incre <- 1
range.cost <- seq(start,end,incre); range.gammma <- seq(start,end,incre)
# range.cost; range.gammma

# Cross Validation Table: (empty for now)
cv.table <- matrix(NA,nrow=length(range.cost),ncol=length(range.gammma))
# cv.table

# Fill in Cross Validation Table:
for (i in 1:length(range.cost)) {
  for (j in 1:length(range.gammma)){
    cv.table[i,j] <- manual.tune(range.cost[[i]],range.gammma[[i]])[[3]]
  }
}
rownames(cv.table) <- range.cost; colnames(cv.table) <- range.gammma
cv.table # Final Output here.
```

```
##           2         3         4
## 2 0.4385246 0.4385246 0.4385246
## 3 0.4385246 0.4385246 0.4385246
## 4 0.4385246 0.4385246 0.4385246
```

```
# 3D Plot:
#scatterplot3d::scatterplot3d(
#  range.cost, xlab = "Cost",
#  range.gammma, ylab = "Gammma",
#  cv.table[,1], main = "Testing Accuracy via Cost and Gammma",
#  pch = 18,
#  col.grid = "purple"
#  )

# Comment:
# We start with Parameters and we set different start and end values as well
# as different increment for tuning. Tuning methods take the following
# procedure:
# 1) Start with a random range;
# 2) Pick one that give the highest;
# 3) Take that coordinate of cost and gamma;
# 4) Decrease range by 1/10 and
#    change increment to 1/10th of the one before.
```

19.2.2 (b) Cross-Validation (Non-Linear)

```
# SVM:
# I wrote a function called manual.tune, that is,
# it is a function allowing me to enter different
# parameters: cost = c, gamma = g:

# Now we change kernal = "radial" instead of linear
manual.tune <- function(c,g){
  ## Apply SVM
  # Ex: c<-1; g<-1
  svm.fit <- svm(
    formula = train[,1] ~.,
    data = train[,-1],
    type = "C-classification",
    kernel = "sigmoid",
    cost=c,
    gamma=g
    )

  ## Now we predict by the model above:
  pred <- predict(
    svm.fit,
    newdata = data.frame(test)
  )

  ## Visualize:
  table <- table(predict=pred, truth=test[,1])
  # roc <- plot.roc(pred, data.test$DRIVER, col="green")

  ## Compute coverage:
  cover.percentile <- sum(diag(table))/sum(table)

  ## Return:
  return(list(
    Summary = summary(svm.fit),
    Table = table,
    Accuracy = cover.percentile))
  }
## End of function

# Ex:
manual.tune(2,1)
```

```
## $Summary
##
## Call:
## svm(formula = train[, 1] ~ ., data = train[, -1], type = "C-classification",
##     kernel = "sigmoid", cost = c, gamma = g)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  sigmoid
##        cost:  2
##       gamma:  1
##      coef.0:  0
##
## Number of Support Vectors:  202
##
##  ( 101 101 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
##
##
##
##
## $Table
##        truth
## predict  -1   1
##      -1  91  23
##      1   20 110
##
## $Accuracy
## [1] 0.8237705
```

```
# Parameters:
start <- 2; end <- 4; incre <- 1
range.cost <- seq(start,end,incre); range.gammma <- seq(start,end,incre)
# range.cost; range.gammma

# Cross Validation Table: (empty for now)
cv.table <- matrix(NA,nrow=length(range.cost),ncol=length(range.gammma))
# cv.table

# Fill in Cross Validation Table:
for (i in 1:length(range.cost)) {
  for (j in 1:length(range.gammma)){
    cv.table[i,j] <- manual.tune(range.cost[[i]],range.gammma[[i]])[[3]]
  }
}
rownames(cv.table) <- range.cost; colnames(cv.table) <- range.gammma
cv.table # Final Output here.
```

```
##           2         3         4
## 2 0.8278689 0.8278689 0.8278689
## 3 0.8237705 0.8237705 0.8237705
## 4 0.8237705 0.8237705 0.8237705
```

```
# 3D Plot:
#scatterplot3d::scatterplot3d(
#  range.cost, xlab = "Cost",
#  range.gammma, ylab = "Gammma",
#  cv.table[,1], main = "Testing Accuracy via Cost and Gammma",
#  pch = 18,
#  col.grid = "purple"
#  )

# Comment:
# We start with Parameters and we set different start and end values as well
# as different increment for tuning. Tuning methods take the following
# procedure:
# 1) Start with a random range;
# 2) Pick one that give the highest;
# 3) Take that coordinate of cost and gamma;
# 4) Decrease range by 1/10 and
#    change increment to 1/10th of the one before.
```

Linear model is the highest. As additional practice, we use linear model to test full dataset.

20 Homework 3

20.1 Problem 1

Ridge Regression and Lasso for Correlated Variables, ISL 6.5.

It is well-known that ridge regression tends to give similar coefficient values to correlated variables, whereas the lasso may give quite different coefficient values to correlated variables. We will not explore this property in a very simple setting.

Suppose that $n = 2$, $p = 2$, $x_{11} = x_{12}$, $x_{21} = x_{22}$. Moreover, suppose that $y_1 + y_2 = 0$ and $x_{11} + x_{21} = 0$ and $x_{12} + x_{22} = 0$, so that the estimate for the intercept in a least squares, ridge regression, or lasso model is zero: $\beta_0 = 0$.

(1). Ridge regression solves the following optimization:

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

In this case, we have $n = 2$, $p = 2$, that is, we have two observations for $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$. Thus, the optimization becomes

$$\underbrace{\min_{\beta} \sum_{i=1}^{2} \left(y_i - \sum_{j=1}^{2} \beta_j x_{i,j} \right)^2}_{\text{RSS of the model}} + \underbrace{\lambda \sum_{j=1}^{2} \beta_j^2}_{l_2 \text{ norm of } \beta \text{, penalty term}}$$

(2). For ridge regression, we have the estimator

$$\hat{\beta}^{\text{ridge}} := (X^T X + \lambda \mathbb{I})^{-1} X^T y$$

Plugging in $x_{11}, x_{12}, x_{21}, x_{22}$, we have

$$\hat{\beta}^{\text{ridge}} := (\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}^T \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} + \lambda \mathbb{I})^{-1} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Expanding the above equation, we have

$$\begin{aligned}\text{Above equation} &= \begin{bmatrix} x_{11}x_{11}+x_{21}x_{21} & x_{11}x_{12}+x_{21}x_{22} \\ x_{12}x_{11}+x_{21}x_{21} & x_{12}x_{12}+x_{22}x_{22}\end{bmatrix} + \begin{bmatrix}\lambda & 0 \\ 0 & \lambda\end{bmatrix} \begin{bmatrix} x_{11}y_1+x_{21}y_2 \\ x_{12}y_1+x_{12}y_2\end{bmatrix} \\ &= \left(\begin{bmatrix} x_{11}x_{11}+x_{21}x_{21} & x_{11}x_{12}+x_{21}x_{22} \\ x_{12}x_{11}+x_{21}x_{21} & x_{12}x_{12}+x_{22}x_{22}\end{bmatrix} + \begin{bmatrix}\lambda & 0 \\ 0 & \lambda\end{bmatrix}\right)\begin{bmatrix}0 \\ 0\end{bmatrix} \\ &= \begin{bmatrix}0 \\ 0\end{bmatrix}\end{aligned}$$

Thus, we achieved $\hat{\beta}_1 = \hat{\beta}_2$.

(3). For lasso regression, we solve the following optimization

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

(4). Let us plug in $x_{11}, x_{12}, x_{21}, x_{22}$ to see what will happen to the solution.

$$\begin{aligned}\beta^{\text{lasso}} &= \arg\min_{\beta} ||y - X\beta||_2^2 + \lambda||\beta||_1 \\ &= \text{sgn}(\beta_j^{\text{LS}})(|\beta_j^{\text{LS}}| - \lambda)^+\end{aligned}$$

Hence, we obtain a family of solution for the coefficient estimator under lasso techinque.

20.2 Problem 2

In this problem, we compare different subset selection methods. We will study the Credit data set, which can be downloaded from canvas site. The data set records balance (average credit card debt) as well as several quantitative predictors: age, cards, education, income, limit, and rating. There are also four qualitative variables: gender, student, status, and ethnicity. We want to fit a regression model of balance on the rest of the variables.

```r
library(ISLR)
data <- read.csv(
  "F://Course/CU Stats/STATS W4241(S) - Statistical Machine Learning/6. Homework/HW3/Credit.csv",
  header = T, sep = ",")
head(data) # Quick view
```

```
##   X  Income Limit Rating Cards Age Education Gender Student Married
## 1 1  14.891  3606    283     2  34        11   Male      No     Yes
## 2 2 106.025  6645    483     3  82        15 Female     Yes     Yes
## 3 3 104.593  7075    514     4  71        11   Male      No      No
## 4 4 148.924  9504    681     3  36        11 Female      No      No
## 5 5  55.882  4897    357     2  68        16   Male      No     Yes
## 6 6  80.180  8047    569     4  77        10   Male      No      No
##   Ethnicity Balance
## 1 Caucasian     333
## 2     Asian     903
## 3     Asian     580
## 4     Asian     964
## 5 Caucasian     331
## 6 Caucasian    1151
```

```r
# Check all names and dimensions:
names(data); dim(data)
```

```
##  [1] "X"         "Income"    "Limit"     "Rating"    "Cards"
##  [6] "Age"       "Education" "Gender"    "Student"   "Married"
## [11] "Ethnicity" "Balance"
```

```
## [1] 400  12
```

```r
sum(is.na(data)) # Good! It's a clean data set.
```

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

```r
library(leaps)
regfit.full <- regsubsets(Balance ~., data)
summary(regfit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(Balance ~ ., data)
## 12 Variables  (and intercept)
##                      Forced in Forced out
## X                        FALSE      FALSE
## Income                   FALSE      FALSE
## Limit                    FALSE      FALSE
## Rating                   FALSE      FALSE
## Cards                    FALSE      FALSE
## Age                      FALSE      FALSE
## Education                FALSE      FALSE
## GenderFemale             FALSE      FALSE
## StudentYes               FALSE      FALSE
## MarriedYes               FALSE      FALSE
## EthnicityAsian           FALSE      FALSE
## EthnicityCaucasian       FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          X   Income Limit Rating Cards Age Education GenderFemale
## 1  ( 1 ) " " " "    " "   "*"    " "   " " " "       " "
## 2  ( 1 ) " " "*"    " "   "*"    " "   " " " "       " "
## 3  ( 1 ) " " "*"    " "   "*"    " "   " " " "       " "
## 4  ( 1 ) " " "*"    "*"   " "    "*"   " " " "       " "
## 5  ( 1 ) " " "*"    "*"   "*"    "*"   " " " "       " "
## 6  ( 1 ) " " "*"    "*"   "*"    "*"   "*" " "       " "
## 7  ( 1 ) " " "*"    "*"   "*"    "*"   "*" " "       "*"
## 8  ( 1 ) "*" "*"    "*"   "*"    "*"   "*" " "       "*"
##          StudentYes MarriedYes EthnicityAsian EthnicityCaucasian
## 1  ( 1 ) " "        " "        " "            " "
## 2  ( 1 ) " "        " "        " "            " "
## 3  ( 1 ) "*"        " "        " "            " "
## 4  ( 1 ) "*"        " "        " "            " "
## 5  ( 1 ) "*"        " "        " "            " "
## 6  ( 1 ) "*"        " "        " "            " "
## 7  ( 1 ) "*"        " "        " "            " "
## 8  ( 1 ) "*"        " "        " "            " "
```

```r
# Comment:
# An asterisk indicates that a given variable is included in the corresponding model.

# Check R-square and RSS:
regfit.full <- regsubsets(Balance ~., data=data, nvmax = 12)
```

```r
reg.summary <- summary(regfit.full)
names(reg.summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```r
reg.summary$rsq # R-square
```

```
##  [1] 0.7458484 0.8751179 0.9498788 0.9535800 0.9541606 0.9546879 0.9548167
##  [8] 0.9549178 0.9549986 0.9550800 0.9551503 0.9552050
```

```r
reg.summary$rss # Residual Sum of Squares
```

```
##  [1] 21435122 10532541  4227219  3915058  3866091  3821620  3810759
##  [8]  3802227  3795415  3788550  3782619  3778009
```

```r
# Plot:
par(mfrow=c(2,2))
plot(
  reg.summary$rss,
  xlab = "Number of Variables",
  ylab = "RSS",
  type = "l"
)
plot(
  reg.summary$adjr2,
  xlab = "Number of Variables",
  ylab = "Adjusted Rsq",
  type = "l"
)
which.max(reg.summary$adjr2)
```

```
## [1] 7
```

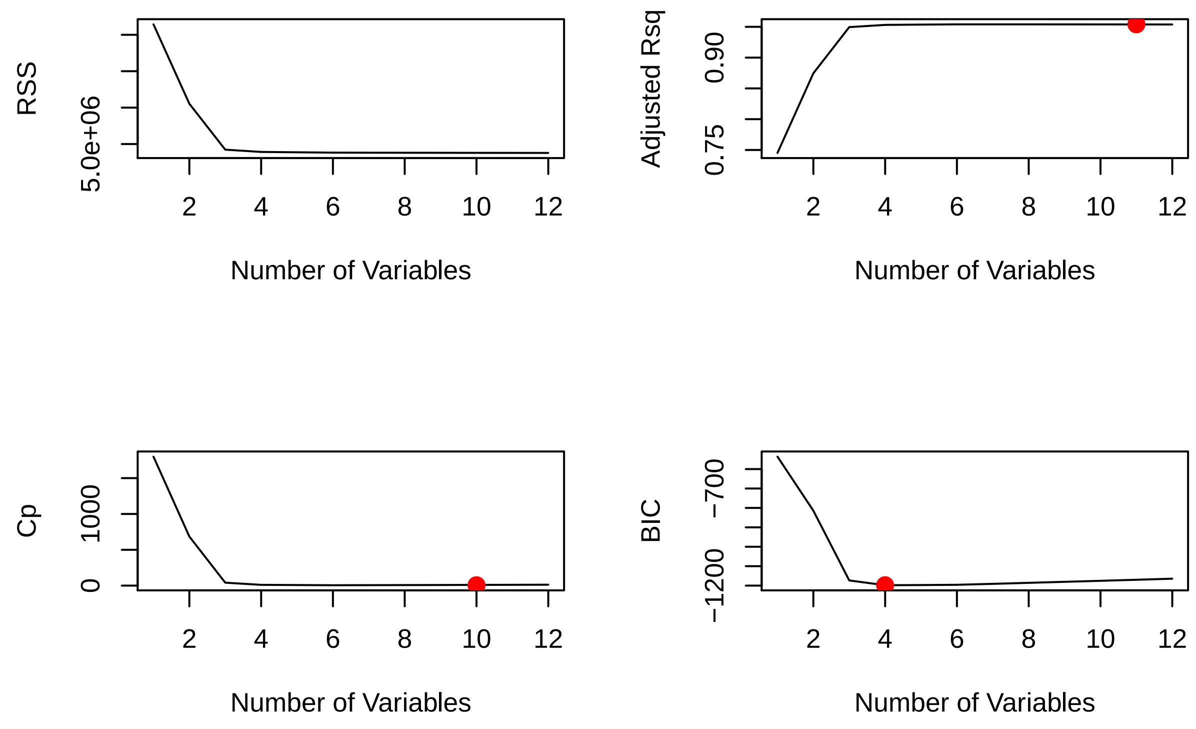
```r
points(11,reg.summary$adjr2[7],
       col = "red", cex = 2, pch = 20)
plot(
  reg.summary$cp,
  xlab = "Number of Variables",
  ylab = "Cp",
  type = 'l'
)
which.min(reg.summary$cp)
```

```
## [1] 6
```

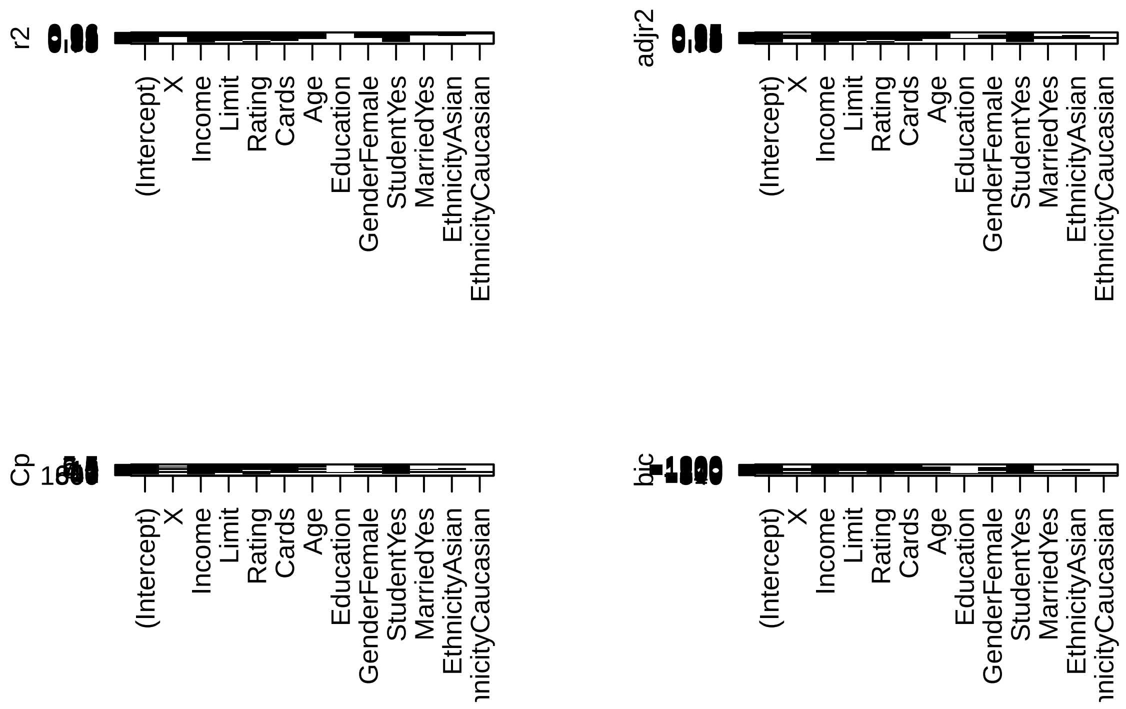
```r
points(10, reg.summary$cp[6],
       col = "red", cex = 2, pch = 20)
which.min(reg.summary$bic)
```

```
## [1] 4
```

```r
plot(
  reg.summary$bic,
  xlab = "Number of Variables",
  ylab = "BIC",
  type = 'l'
)
points(4, reg.summary$bic[4],
       col = "red", cex = 2, pch = 20)
```



```
plot(regfit.full, scale = "r2")
plot(regfit.full, scale = "adjr2")
plot(regfit.full, scale = "Cp")
plot(regfit.full, scale = "bic")
```



```r
coef(regfit.full, 4)
```

```
##   (Intercept)        Income         Limit         Cards    StudentYes
## -499.7272117   -7.8392288    0.2666445   23.1753794   429.6064203
```

```r
# Use regsubsets() function to perform
# forward stepwise or backward stepwise selection.
regfit.fwd <- regsubsets(
  Balance ~.,
  data = data,
  nvmax = 12,
  method = "forward")
summary(regfit.fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(Balance ~ ., data = data, nvmax = 12, method = "forward")
## 12 Variables  (and intercept)
##                    Forced in Forced out
## X                      FALSE      FALSE
## Income                 FALSE      FALSE
## Limit                  FALSE      FALSE
## Rating                 FALSE      FALSE
## Cards                  FALSE      FALSE
## Age                    FALSE      FALSE
## Education              FALSE      FALSE
## GenderFemale           FALSE      FALSE
## StudentYes             FALSE      FALSE
## MarriedYes               FALSE      FALSE
## EthnicityAsian           FALSE      FALSE
## EthnicityCaucasian       FALSE      FALSE
## 1 subsets of each size up to 12
## Selection Algorithm: forward
##           X   Income Limit Rating Cards Age Education GenderFemale
## 1  ( 1 )  " " " "    " "   "*"    " "   " " " "       " "
## 2  ( 1 )  " " "*"    " "   "*"    " "   " " " "       " "
## 3  ( 1 )  " " "*"    " "   "*"    " "   " " " "       " "
## 4  ( 1 )  " " "*"    "*"   "*"    " "   " " " "       " "
## 5  ( 1 )  " " "*"    "*"   "*"    "*"   " " " "       " "
## 6  ( 1 )  " " "*"    "*"   "*"    "*"   "*" " "       " "
## 7  ( 1 )  " " "*"    "*"   "*"    "*"   "*" " "       "*"
## 8  ( 1 )  "*" "*"    "*"   "*"    "*"   "*" " "       "*"
## 9  ( 1 )  "*" "*"    "*"   "*"    "*"   "*" " "       "*"
## 10  ( 1 ) "*" "*"    "*"   "*"    "*"   "*" " "       "*"
## 11  ( 1 ) "*" "*"    "*"   "*"    "*"   "*" " "       "*"
## 12  ( 1 ) "*" "*"    "*"   "*"    "*"   "*" "*"       "*"
##           StudentYes MarriedYes EthnicityAsian EthnicityCaucasian
## 1  ( 1 )  " "        " "        " "            " "
## 2  ( 1 )  " "        " "        " "            " "
## 3  ( 1 )  "*"        " "        " "            " "
## 4  ( 1 )  "*"        " "        " "            " "
## 5  ( 1 )  "*"        " "        " "            " "
## 6  ( 1 )  "*"        " "        " "            " "
## 7  ( 1 )  "*"        " "        " "            " "
## 8  ( 1 )  "*"        " "        " "            " "
## 9  ( 1 )  "*"        " "        "*"            " "
## 10  ( 1 ) "*"        "*"        "*"            " "
## 11  ( 1 ) "*"        "*"        "*"            "*"
## 12  ( 1 ) "*"        "*"        "*"            "*"
```

```r
regfit.bwd <- regsubsets(
  Balance ~.,
  data = data, nvmax = 12,
  method = "backward"
)
summary(regfit.bwd)
```

```
## Subset selection object
## Call: regsubsets.formula(Balance ~ ., data = data, nvmax = 12, method = "backward")
## 12 Variables  (and intercept)
##                      Forced in Forced out
## X                        FALSE      FALSE
## Income                   FALSE      FALSE
## Limit                    FALSE      FALSE
## Rating                   FALSE      FALSE
## Cards                    FALSE      FALSE
## Age                      FALSE      FALSE
## Education                FALSE      FALSE
## GenderFemale             FALSE      FALSE
## StudentYes               FALSE      FALSE
## MarriedYes               FALSE      FALSE
## EthnicityAsian           FALSE      FALSE
## EthnicityCaucasian       FALSE      FALSE
## 1 subsets of each size up to 12
## Selection Algorithm: backward
##           X   Income Limit Rating Cards Age Education GenderFemale
## 1  ( 1 )  " " " "    "*"   " "    " "   " " " "       " "
## 2  ( 1 )  " " "*"    "*"   " "    " "   " " " "       " "
## 3  ( 1 )  " " "*"    "*"   " "    " "   " " " "       " "
## 4  ( 1 )  " " "*"    "*"   " "    "*"   " " " "       " "
## 5  ( 1 )  " " "*"    "*"   "*"    "*"   " " " "       " "
## 6  ( 1 )  " " "*"    "*"   "*"    "*"   "*" " "       " "
## 7  ( 1 )  " " "*"    "*"   "*"    "*"   "*" " "       "*"
## 8  ( 1 )  "*" "*"    "*"   "*"    "*"   "*" " "       "*"
## 9  ( 1 )  "*" "*"    "*"   "*"    "*"   "*" " "       "*"
## 10  ( 1 ) "*" "*"    "*"   "*"    "*"   "*" " "       "*"
## 11  ( 1 ) "*" "*"    "*"   "*"    "*"   "*" " "       "*"
## 12  ( 1 ) "*" "*"    "*"   "*"    "*"   "*" "*"       "*"
##           StudentYes MarriedYes EthnicityAsian EthnicityCaucasian
## 1  ( 1 )  " "        " "        " "            " "
## 2  ( 1 )  " "        " "        " "            " "
## 3  ( 1 )  "*"        " "        " "            " "
## 4  ( 1 )  "*"        " "        " "            " "
## 5  ( 1 )  "*"        " "        " "            " "
## 6  ( 1 )  "*"        " "        " "            " "
## 7  ( 1 )  "*"        " "        " "            " "
## 8  ( 1 )  "*"        " "        " "            " "
## 9  ( 1 )  "*"        " "        "*"            " "
## 10  ( 1 ) "*"        "*"        "*"            " "
## 11  ( 1 ) "*"        "*"        "*"            "*"
## 12  ( 1 ) "*"        "*"        "*"            "*"
```

```
# Check the best nine-variable:
coef(regfit.full, 9)
```

```
##    (Intercept)              X         Income          Limit         Rating
##  -501.11909712     0.04233333    -7.81283276     0.19176507     1.12362322
##          Cards            Age   GenderFemale     StudentYes EthnicityAsian
##    18.07910749    -0.62198701    -9.51102994   426.37051557     9.54024975
```

```
coef(regfit.fwd, 9)
```

```
##    (Intercept)              X         Income          Limit         Rating
##  -501.11909712     0.04233333    -7.81283276     0.19176507     1.12362322
##          Cards            Age   GenderFemale     StudentYes EthnicityAsian
##    18.07910749    -0.62198701    -9.51102994   426.37051557     9.54024975
```

```
coef(regfit.bwd, 9)
```

```
##    (Intercept)              X         Income          Limit         Rating
##  -501.11909712     0.04233333    -7.81283276     0.19176507     1.12362322
##          Cards            Age   GenderFemale     StudentYes EthnicityAsian
##    18.07910749    -0.62198701    -9.51102994   426.37051557     9.54024975
```

We saw it is possible to choose among a set of models of different sizes using C_p, BIC, and adjusted R^2. We now consider how to do this using validation set and cross-validation approaches.

In order for these approaches to yield accurate estiamtes of the test error, we must use only the training observations to perform all aspects of model-fitting — including variable selection. Therefore, the determination of which model of a given size is best must be made using only the training observation.

```r
# Split training and testing set:
set.seed(1)
train <- sample(c(TRUE, FALSE),
                nrow(data),
                rep = TRUE)
test <- (!train)

# Training and Errors:
regfit.best <- regsubsets(
  Balance ~.,
  data = data[train,],
  nvmax = 12
)
test.mat <- model.matrix(
  Balance ~.,
  data = data[test,])
val.errors <- rep(NA, 10)
for (i in 1:12) {
  coefi <- coef(regfit.best, id = i)
  pred <- test.mat[,names(coefi)] %*% coefi
  val.errors[i] <- mean((data$Balance[test] - pred)^2)
}
val.errors
```

```
##  [1] 51754.40 24326.78 12905.05 11616.12 11582.29 12176.73 12041.24
##  [8] 12024.54 11980.66 11930.86 11943.03 11944.39
```

```r
which.min(val.errors) # 5
```

```
## [1] 5
```

```r
coef(regfit.best, 5)
```

```
##  (Intercept)       Income        Limit        Cards          Age
## -443.5105267   -7.8334235    0.2659310   22.0262551   -0.8361878
##   StudentYes
##  454.7860565
```

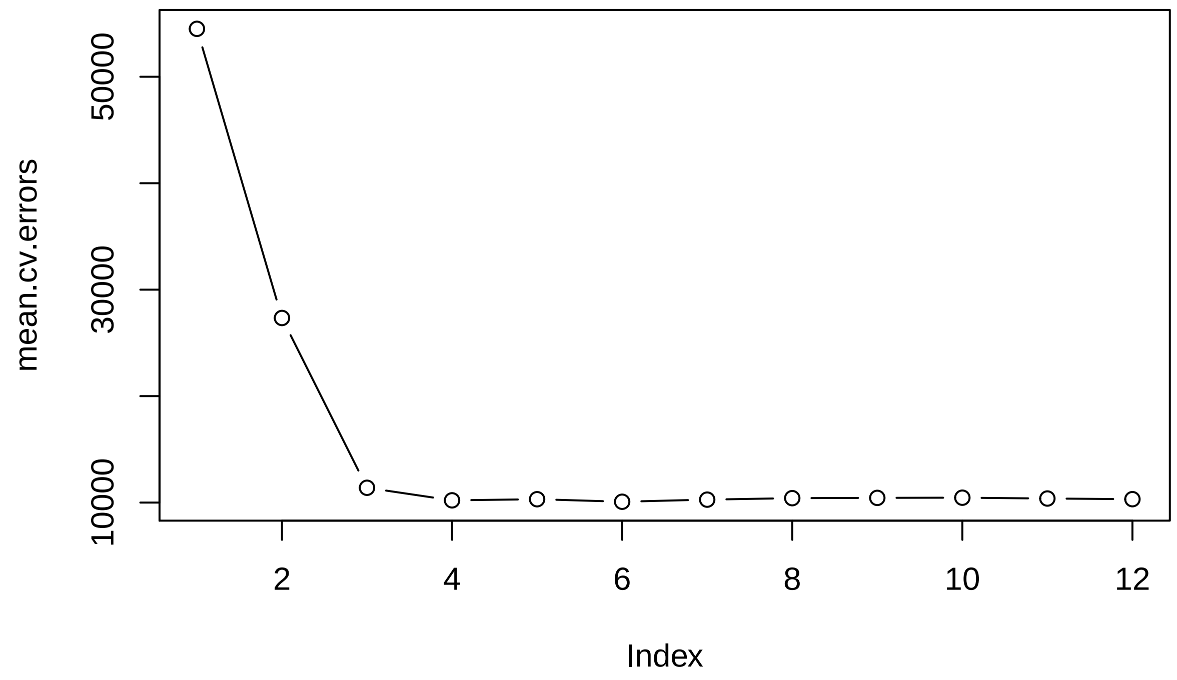
```r
# Now we write our own predict function:
predict.regsubsets <- function(object, newdata, id, ...){
  form <- as.formula(object$call[[2]])
  mat <- model.matrix(form, newdata)
  coefi <- coef(object, id = id)
  xvars <- names(coefi)
  mat[,xvars]%*%coefi
}

# Finally, perform best subset selection
regfit.best <- regsubsets(
  Balance ~.,
  data = data, nvmax = 12
)
coef(regfit.best, 12)
```

```
##       (Intercept)                  X             Income
##      -487.07423743         0.04104764        -7.80739871
##              Limit            Rating             Cards
##         0.19052127        1.14248766       17.83638753
##                Age         Education      GenderFemale
##        -0.62954679       -1.09830902       -9.54615446
##         StudentYes        MarriedYes    EthnicityAsian
##       426.16715394       -8.78055030       16.85751762
## EthnicityCaucasian
##         9.29289272
```

```
# K-fold CV:
k <- 10
set.seed(1)
folds <- sample(1:k, nrow(data), replace = TRUE)
cv.errors <- matrix(NA, k, 12,
                    dimnames = list(NULL, paste(1:12)))
for (j in 1:k){
  best.fit <- regsubsets(Balance ~.,
                         data = data[folds!=j,], nvmax = 12)
  for (i in 1:12){
    pred <- predict(best.fit, data[folds == j,], id = i)
    cv.errors[j,i] <- mean(
      (data$Balance[folds == j] - pred)^2
    )
  }
}

# Results:
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors
```

```
##        1        2        3        4        5        6        7        8
## 54501.60 27339.31 11389.85 10208.96 10314.11 10077.26 10275.25 10413.53
##        9       10       11       12
## 10440.84 10458.38 10379.73 10318.31
```

```
# Plot:
par(mfrow=c(1,1))
plot(mean.cv.errors, type = 'b')
```



```
# Subset selection on selected variables:
reg.best <- regsubsets(
  Balance~.,
  data = data,
  nvmax = 12
)
coef(reg.best, 12)
```

```
##       (Intercept)                 X            Income
##     -487.07423743        0.04104764       -7.80739871
##             Limit            Rating             Cards
##        0.19052127        1.14248766       17.83638753
##               Age         Education      GenderFemale
##       -0.62954679       -1.09830902       -9.54615446
##        StudentYes        MarriedYes    EthnicityAsian
##      426.16715394       -8.78055030       16.85751762
## EthnicityCaucasian
##        9.29289272
```

We will use glmnet package in order to perform ridge regression and lasso.

```
# Set up data:
x <- model.matrix(Balance~., data)[,-1]
y <- data$Balance

# Ridge Regression
library(glmnet)
```

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

```
## [1]  13 100
```

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

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

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

```
##        (Intercept)                  X             Income
##       4.437406e+02       7.098518e-04       2.072944e-01
##              Limit             Rating              Cards
##       6.255481e-03       9.352902e-02       1.094867e+00
##                Age          Education       GenderFemale
##      -7.433736e-03      -3.648999e-02       7.279655e-01
##         StudentYes         MarriedYes     EthnicityAsian
##       1.522470e+01      -2.740740e-01      -3.232180e-01
## EthnicityCaucasian
##      -8.954612e-02
```

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

```
## [1] 157.2132
```

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

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

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

```
##        (Intercept)                  X             Income
##        10.95570094         0.00132288         0.47042764
##              Limit             Rating              Cards
##         0.04709662         0.70327881        10.00914897
##                Age          Education       GenderFemale
##        -0.54080663         0.01398148         4.63069943
##         StudentYes         MarriedYes     EthnicityAsian
##       156.69520888        -6.12532307         0.62575951
## EthnicityCaucasian
##         1.42887731
```

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

```
## [1] 157.2132
```

```r
# Comment:
# Notice that much larger l2 norm of the coefficients associated
# with this smaller value of lambda.

# Predict
predict(ridge.mod,
        s=50,
        type = "coefficients")[1:10,]
```

```
##   (Intercept)             X        Income         Limit        Rating
## -387.01506763    0.02788993   -4.71033241    0.11026631    1.60846689
##          Cards           Age     Education  GenderFemale    StudentYes
##   16.10495492   -1.00880354   -0.40547730   -3.16706653  372.95067139
```

```
# Now we want to estimate the test error
# Split training and testing
set.seed(1)
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
y.test = y[test]

# Fit ridge:
ridge.mod <- glmnet(x[train,],
                    y[train],
                    alpha=0,
                    lambda=grid,
                    thresh = 1e-12)
ridge.pred <- predict(ridge.mod,s=4,newx = x[test,])
mean((ridge.pred - y.test)^2)
```

```
## [1] 10629.73
```

```
# Comment:
# This gives us the MSE for test set.

# We could predict each test observation
# using mean of the training observations.
mean((mean(y[test]) - y.test)^2)
```

```
## [1] 192298.3
```

```
# We could also get the same result by
# fitting a ridge regression model with
# a large lambda
ridge.pred <- predict(ridge.mod,
                      s = 1e10,
                      newx = x[test,])
mean((ridge.pred - y.test)^2)
```

```
## [1] 194734.7
```

```
# Check least squaers
# which is ridge with lambda = 0
ridge.pred <- predict(ridge.mod,
                      s = 0,
                      newx = x[test,],
                      exact = T)
mean((ridge.pred - y.test)^2)
```

```
## [1] 10743.51
```

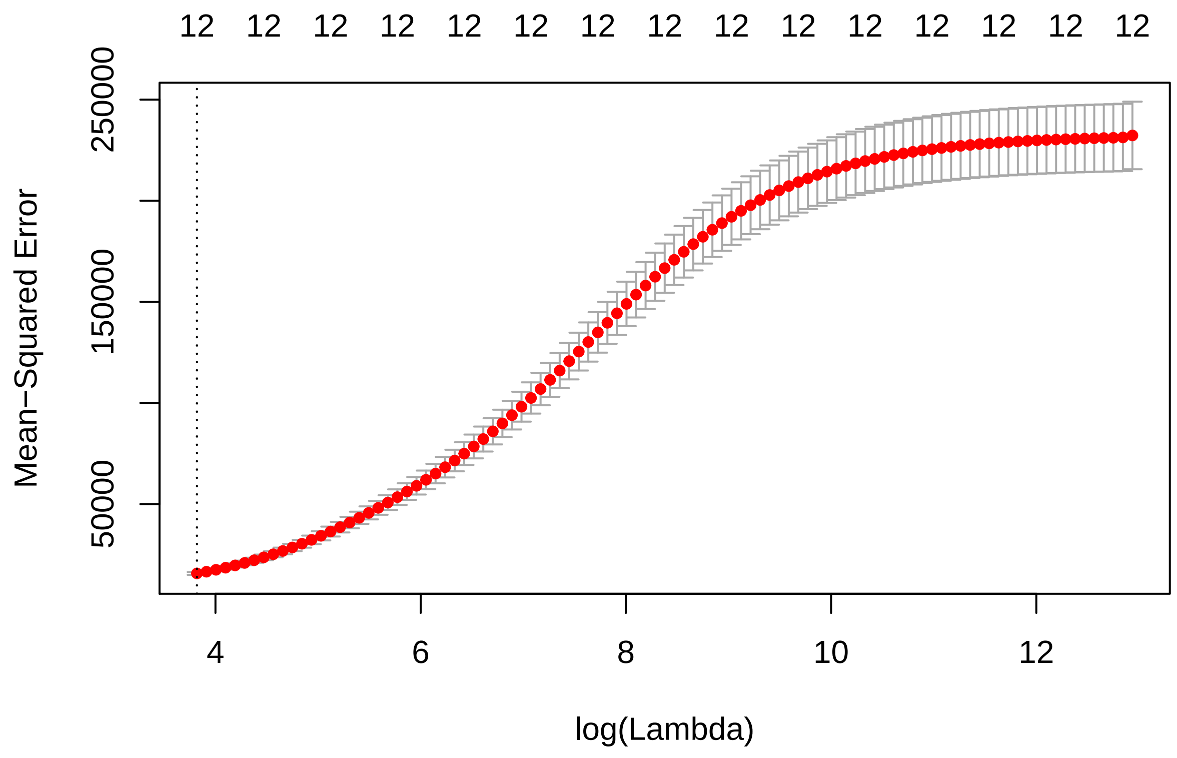
```
lm(y ~ x, subset = train)
```

```
##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
##         (Intercept)                   xX              xIncome
##           -514.25138              0.05894              -7.89399
##               xLimit              xRating                xCards
##              0.20960              0.93521              21.77308
##                 xAge           xEducation         xGenderFemale
##             -0.62060             -1.73676             -18.42083
##          xStudentYes          xMarriedYes      xEthnicityAsian
##            438.05260            -11.30673              30.59576
## xEthnicityCaucasian
##             21.48476
```

```
predict(ridge.mod, s = 0,
        exact = T, type = "coefficients")[1:12,]
```

```
##     (Intercept)               X          Income           Limit          Rating
##   -514.26738239      0.05893909     -7.89395739      0.20954177      0.93601568
##           Cards             Age       Education    GenderFemale      StudentYes
##     21.76904813     -0.62062602     -1.73681038    -18.42100572    438.04771338
##      MarriedYes  EthnicityAsian
##    -11.30935593     30.59747319
```

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



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

```
## [1] 45.57503
```

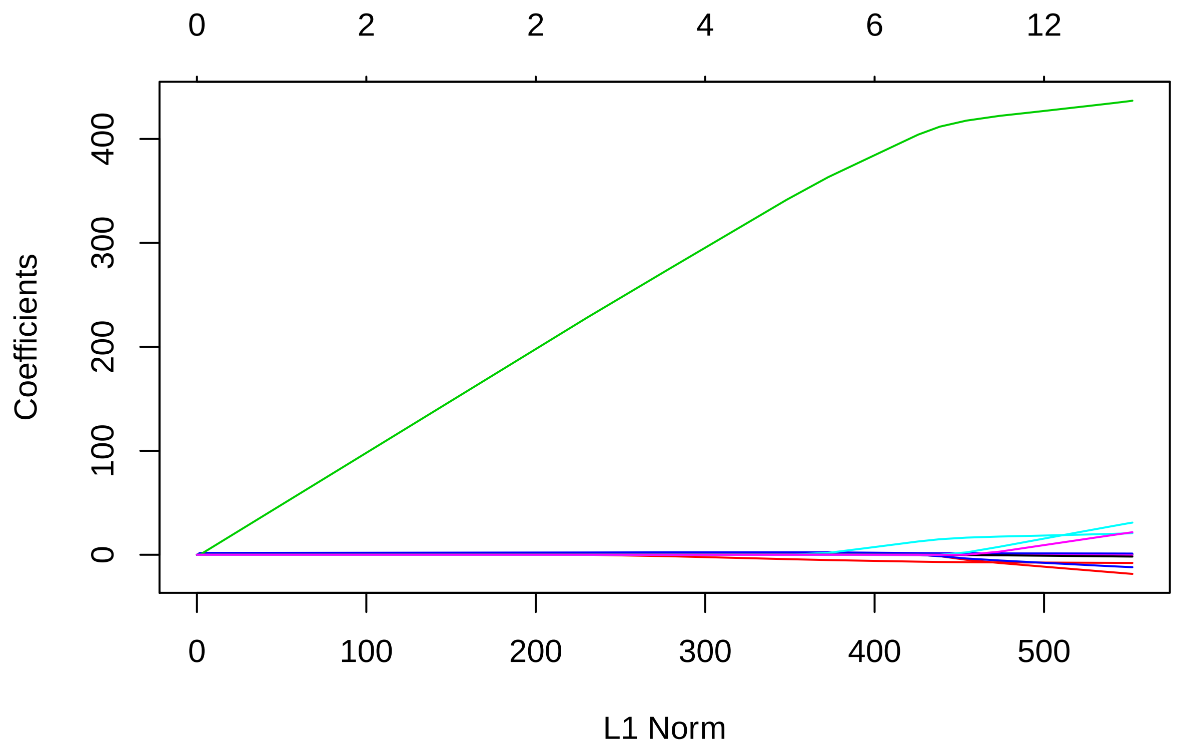
```
# How about test MSE?
ridge.pred <- predict(ridge.mod,
                      s = bestlam, newx = x[test,])
mean((ridge.pred - y.test)^2)
```

```
## [1] 14972.84
```

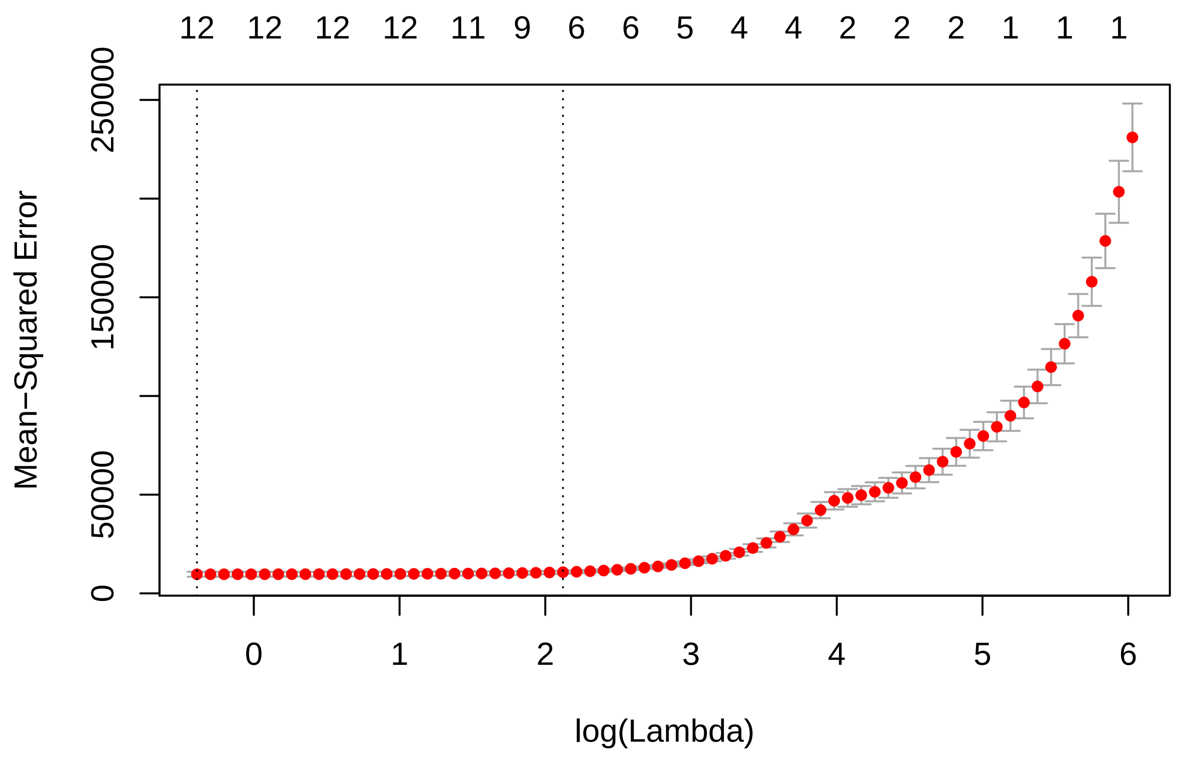
```
out <- glmnet(x, y, alpha = 0)
predict(out,
        type = "coefficients",
        s = bestlam)[1:12,]
```

```
##    (Intercept)             X        Income         Limit        Rating
##  -394.95250417    0.02879516   -4.89652995    0.11194699    1.62982781
##          Cards           Age     Education  GenderFemale    StudentYes
##    16.03371853   -0.99357568   -0.43248752   -3.53888279  376.69203127
##     MarriedYes EthnicityAsian
##   -12.36118029   12.64500063
```

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



```
# Comment:
# We see the coefficient plot that depending
# on the choise of tuning parameter.

# CV and Test error:
set.seed(1)
cv.out <- cv.glmnet(x[train,],
                    y[train], alpha = 1)
plot(cv.out)
```



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

```
## [1] 10600.11
```

```
# Comment:
# This is substantially lower than the test
# set MSE of the null model and of
# least squares.

# Lasso model with lambda chosen
out <- glmnet(x, y, alpha = 1, lambda =  grid)
lasso.coef <- predict(
  out, type = "coefficients",
  s = bestlam)[1:12,]
lasso.coef
```

```
##   (Intercept)             X        Income         Limit        Rating
##  -490.84088807    0.03534749   -7.71554696    0.17366046    1.37247069
##         Cards           Age     Education  GenderFemale    StudentYes
##    16.29162828   -0.60623545   -0.82574926   -8.12111664  422.74904847
##    MarriedYes EthnicityAsian
##    -7.72139308   13.29553015
```

```
lasso.coef[lasso.coef!=0]
```

```
##   (Intercept)             X        Income         Limit        Rating
##  -490.84088807    0.03534749   -7.71554696    0.17366046    1.37247069
##         Cards           Age     Education  GenderFemale    StudentYes
##    16.29162828   -0.60623545   -0.82574926   -8.12111664  422.74904847
##    MarriedYes EthnicityAsian
##    -7.72139308   13.29553015
```

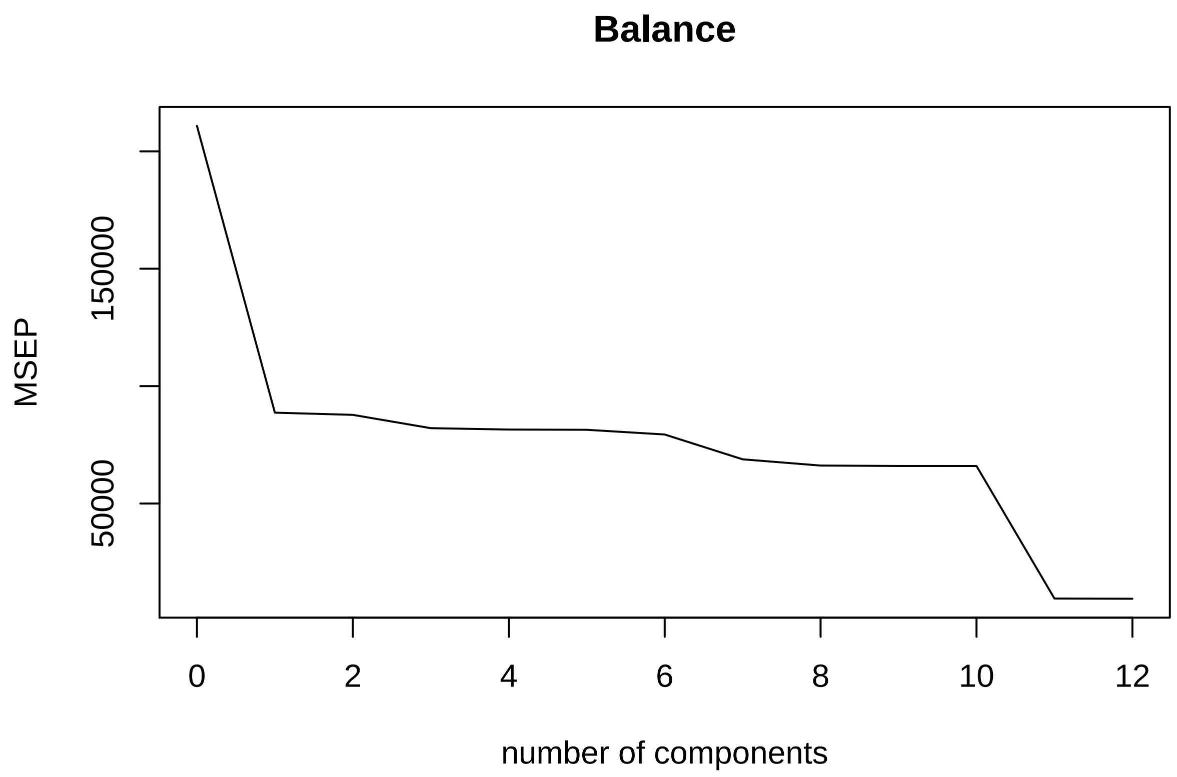
The following is optional for this homework:

Now we attempt Principal Components Regression.

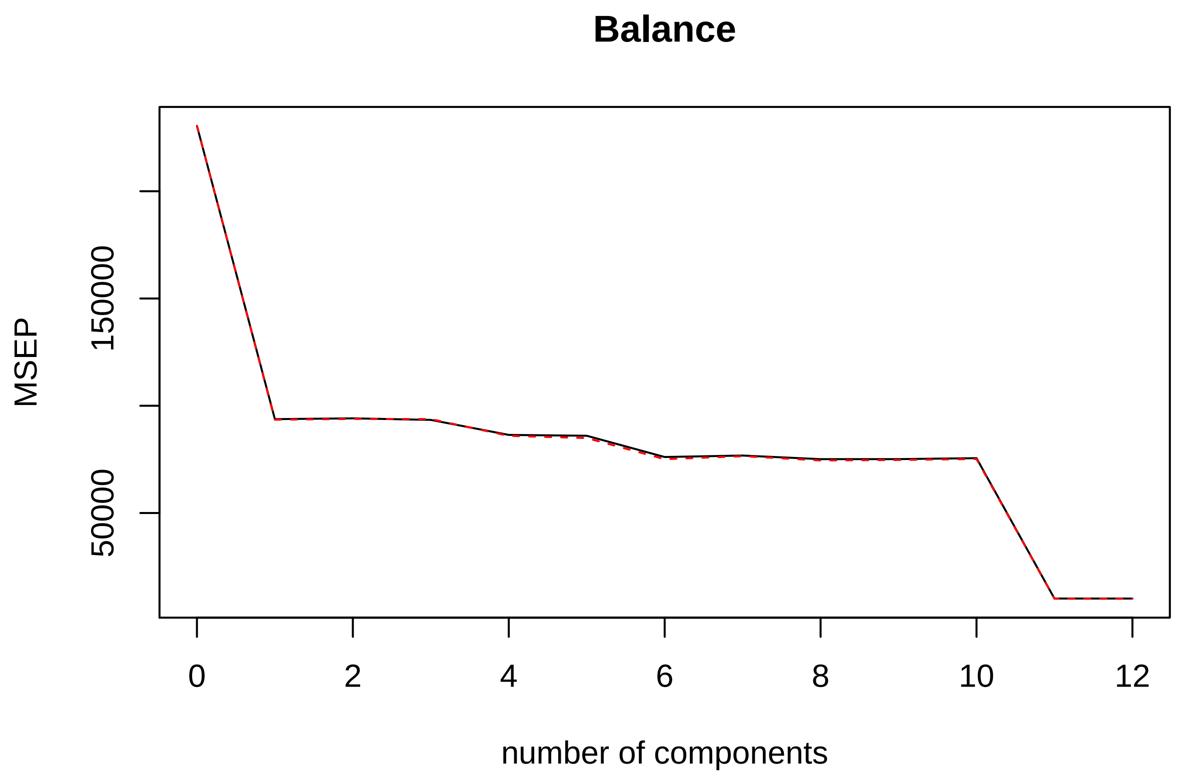
```
library(pls)
set.seed(2)
pcr.fit <- pcr(
  Balance ~.,
  data = data, scale = TRUE,TRUEvalidation = "CV"
)
summary(pcr.fit)
```

```
## Data:    X dimension: 400 12
##  Y dimension: 400 1
## Fit method: svdpc
## Number of components considered: 12
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X          22.98    36.54    46.05    55.25    64.23    72.34    80.36
## Balance    57.93    58.37    61.06    61.34    61.39    62.34    67.36
##          8 comps  9 comps  10 comps  11 comps  12 comps
## X          87.75    94.43     97.80     99.98    100.00
## Balance    68.62    68.70     68.71     95.49     95.52
```

```
validationplot(pcr.fit, val.type = "MSEP")
```



```r
# Comment:
# We see that the smallest CV error occurs
# when M = 11 components are used. This is
# barely fewer than M = 12, which amounts
# to simply performing least squares, because
# when all of the components are used in PCR
# no dimension reduction occurs. However,
# from the plot we also see that the CV error
# is roughly the same when only one component
# is included in the model. This suggests
# that a model that uses just a small
# number of components might suffice.

# Perform PCR on training data:
set.seed(1)
pcr.fit <- pcr(
  Balance ~.,
  data = data, subset = train,
  scale = TRUE,
  validation = "CV"
)
validationplot(pcr.fit, val.type = "MSEP")
```



```
pcr.pred <- predict(pcr.fit,x[test,],ncomp = 10)
mean((pcr.pred - y.test)^2)
```

```
## [1] 67014.07
```

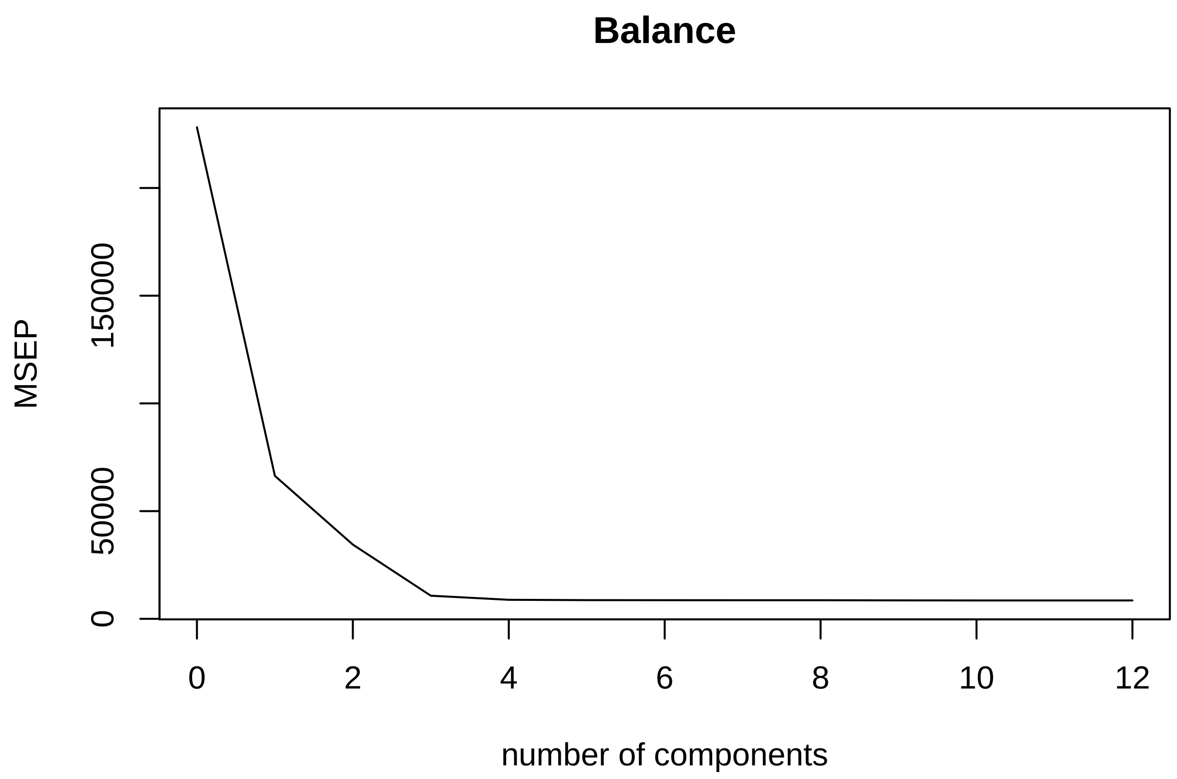
```
# Fit PCR on full data set:
pcr.fit <- pcr(y ~ x,
               scale = TRUE, ncomp = 7)
summary(pcr.fit)
```

```
## Data:    X dimension: 400 12
##  Y dimension: 400 1
## Fit method: svdpc
## Number of components considered: 7
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X    22.98    36.54    46.05    55.25    64.23    72.34    80.36
## y    57.93    58.37    61.06    61.34    61.39    62.34    67.36
```

```
# Partial Least Squares:
set.seed(1)
pls.fit = plsr(
  Balance ~., data = data,
  subset = train, scale = TRUE,TRUEvalidation = "CV"
)
summary(pls.fit)
```

```
## Data:    X dimension: 200 12
##  Y dimension: 200 1
## Fit method: kernelpls
## Number of components considered: 12
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X          23.41    31.68    35.93    46.64    55.75    61.85    68.53
## Balance    70.92    84.90    95.30    96.13    96.20    96.21    96.21
##          8 comps  9 comps  10 comps  11 comps  12 comps
## X          77.28    80.13     85.75     93.52    100.00
## Balance    96.21    96.25     96.26     96.26     96.26
```

```
validationplot(pls.fit, val.type = "MSEP")
```



```
# Test set MSE:
pls.pred <- predict(pls.fit, x[test,],ncomp = 2)
mean((pls.pred - y.test)^2)
```

```
## [1] 34724.48
```

```
# Perform PLS using full data set:
pls.fit <- plsr(
  Balance ~., data = data, scale = TRUE, ncomp = 2
)
summary(pls.fit)
```

```
## Data:    X dimension: 400 12
##  Y dimension: 400 1
## Fit method: kernelpls
## Number of components considered: 2
## TRAINING: % variance explained
##          1 comps  2 comps
## X          22.54    29.96
## Balance    69.67    86.42
```