ECO3080 Machine Learning for Business

Tutorial Notes with R

Long (Neo) Ma

2023/08/13

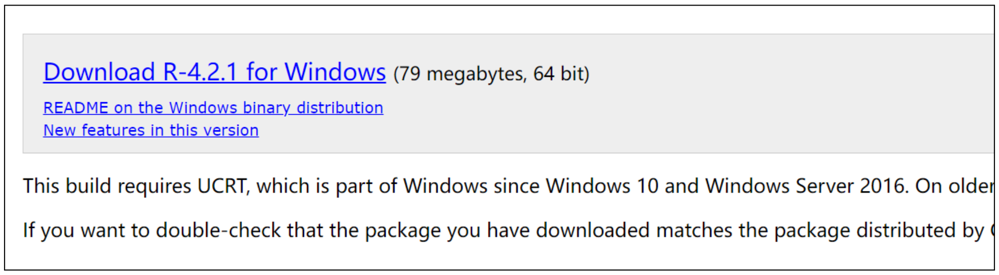
Contents

Chapter 1

Starting with R

1.1 Installation of R and R-Studio

- R is a free software environment for statistical computing and graphics. It can compile and run on Windows and MacOS. Please download R at www.r-project.org.
  - Choose proper CRAN mirror, e.g. "Tsinghua University".
  - Choose proper version, e.g. "Download R for Windows".
  - Choose "base" (which is enough for beginners) and download it.



Download R-4.2.1 for Windows (79 megabytes, 64 bit)
README on the Windows binary distribution
New features in this version

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older

If you want to double-check that the package you have downloaded matches the package distributed by

- IDE is the abbreviation of "Integrated Development Environment" where you can input the codes (or commands) and obtain the outcomes (or results). Here are three possible choices for you:
  1. R Gui (Not recommended)
  2. R-Studio (Very good but only for R)
  3. Visual Studio Code (Not only for R, but for Python, C++, etc. as well)
- Please do not use Python or other languages in this course.

1.2 Basic Functions

- The first thing you should do is to set your workspace

```
## Set your workspace
getwd()                           # your current working directory
setwd("C:/Users/Neoma/Desktop") # change the directory to Desktop
```

- If you have any problems with a specific command, you can refer to its help file.

```
## Getting Help in R
help("lm")
```

- In R, you always use packages designed by engineers or scholars. Before using them, you need to install and load them by the following commands.

```
## Install Packages in R
install.packages("ggplot2")
library(ggplot2)                  # load it before using it
update.packages("ggplot2")        # update the package
```

- You may use the following functions very often.

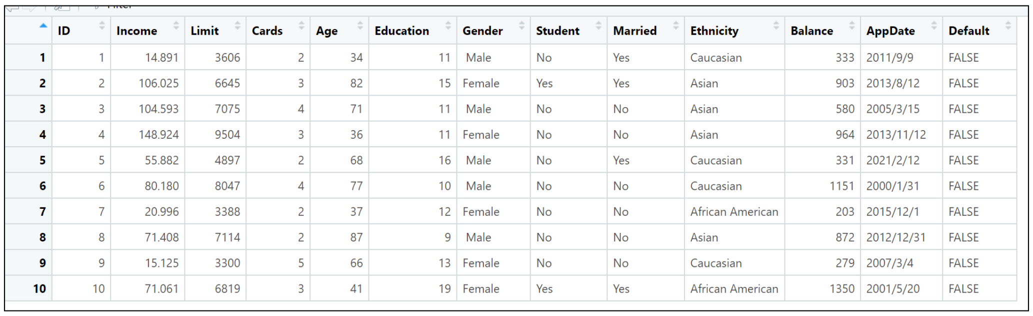
```
## Functions for Math and Statistics
x <- 4.125                  # x is a scalar
print(abs(x))               # absolute value of x
print(sqrt(x))              # square root of x
print(ceiling(x))           # the smallest int >= x
print(floor(x))             # the biggest int <= x
print(log(x))               # natural log of x
print(exp(x))               # exponential of x

y <- c(1, 2, 3, 4, 5)       # y is a vector
print(length(y))            # number of elements in y
print(mean(y))              # mean of y
print(median(y))            # median of y
print(sd(y))                # standard deviation of y
print(var(y))               # variance of y
print(sum(y))               # sum of y
print(diff(y, lag = 1))     # difference of y
print(min(y))               # minimum of y
print(max(y))               # maximum of y

rep(1:2, 5)                 # replicate 1 to 2 five times
set.seed(520)               # set random seed
```

1.3 Basic Data Management

- The most important data structure used in this course is data frame. The easiest way to set up a data frame is to import a dataset from outside. The following example shows you how to import a ".csv" file (and do not transform the strings into factors).

```r
## Import .csv Data by read.table()
credit_data <- read.table("RawCredit.csv", header = TRUE,
                          sep = ",", stringsAsFactors = FALSE)
```

- After importing the data, there are 13 variables (columns) and 10 observations (rows). Obviously, some variables are numerical variables, e.g. Income, Limit, etc., while others are strings, e.g. Gender and Ethnicity. This tells you that different variables may have different types.



	ID	Income	Limit	Cards	Age	Education	Gender	Student	Married	Ethnicity	Balance	AppDate	Default
1	1	14.891	3606	2	34	11	Male	No	Yes	Caucasian	333	2011/9/9	FALSE
2	2	106.025	6645	3	82	15	Female	Yes	Yes	Asian	903	2013/8/12	FALSE
3	3	104.593	7075	4	71	11	Male	No	No	Asian	580	2005/3/15	FALSE
4	4	148.924	9504	3	36	11	Female	No	No	Asian	964	2013/11/12	FALSE
5	5	55.882	4897	2	68	16	Male	No	Yes	Caucasian	331	2021/2/12	FALSE
6	6	80.180	8047	4	77	10	Male	No	No	Caucasian	1151	2000/1/31	FALSE
7	7	20.996	3388	2	37	12	Female	No	No	African American	203	2015/12/1	FALSE
8	8	71.408	7114	2	87	9	Male	No	No	Asian	872	2012/12/31	FALSE
9	9	15.125	3300	5	66	13	Female	No	No	Caucasian	279	2007/3/4	FALSE
10	10	71.061	6819	3	41	19	Female	Yes	Yes	African American	1350	2001/5/20	FALSE

- The following codes teach you how to check the type of variables and change them into the type that you want, say, from strings to factors (or to date).

```r
## Check the Type
print(is.numeric(credit_data$Rating))
print(is.character(credit_data$Rating))
print(is.data.frame(credit_data))

## String to Factor
credit_data$Gender <- as.factor(credit_data$Gender)
credit_data$Student <- as.factor(credit_data$Student)
credit_data$Married <- factor(credit_data$Married, order = F)
credit_data$Ethnicity <- factor(credit_data$Ethnicity, order = F)

## String to Date
myformat <- "%Y/%m/%d"    # %Y = "xxxx", %y = "xx"
credit_data$AppDate <- as.Date(credit_data$AppDate, myformat)

## Check Types of All Variables
str(credit_data)
```

- Sometimes, you may want to change the name of variables. For example, you want to rename the first column of credit_data as "UserID", and rename the second and third columns as "UserIncome" and "UserLimit", respectively.

```r
## Rename your variable
names(credit_data)[1] <- "UserID"
names(credit_data)[2:3] <- c("UserIncome", "UserLimit")
print(names(credit_data))
```

- Sometimes, you may also want to re-code some variables. For example, variable "Rating" indicates the credit score of users. If you want to reconstruct this variable by the following criterion, you can use the codes below.
  - If "Rating" ¿= 500, "Rating" = Good
  - If 200 ¡ "Rating" ¡ 500, "Rating" = Medium
  - If "Rating" ¡= 200, "Rating" = Bad

```r
## Re-code Rating
credit_data$Rating[credit_data_final$Rating >= 500] <- "Good"
credit_data$Rating[credit_data_final$Rating < 500 &
                   credit_data_final$Rating > 200] <- "Medium"
credit_data$Rating[credit_data_final$Rating <= 200] <- "Bad"
```

- Sometimes, you may also want to include more variables or more observations into your original data frame. The following codes provide you with some examples.

```r
## Add Column
credit_data_addcol <- read.table("CreditAddCol.csv", header = T,
                                  sep = ",", stringsAsFactors = F)

# (1) use merge command (you need a key to link them)
credit_data_1 <- merge(credit_data, credit_data_addcol,
                        by = "UserID")
# (2) use cbind command (two data sets with same number of rows)
credit_data_2 <- cbind(credit_data, credit_data_addcol)

## Add Row
credit_data_addrow <- read.table("CreditAddRow.csv", header = T,
                                  sep = ",", stringsAsFactors = F)
# use rbind command (two data sets with same set of variables)
credit_data_final <- rbind(credit_data_1, credit_data_addrow)
```

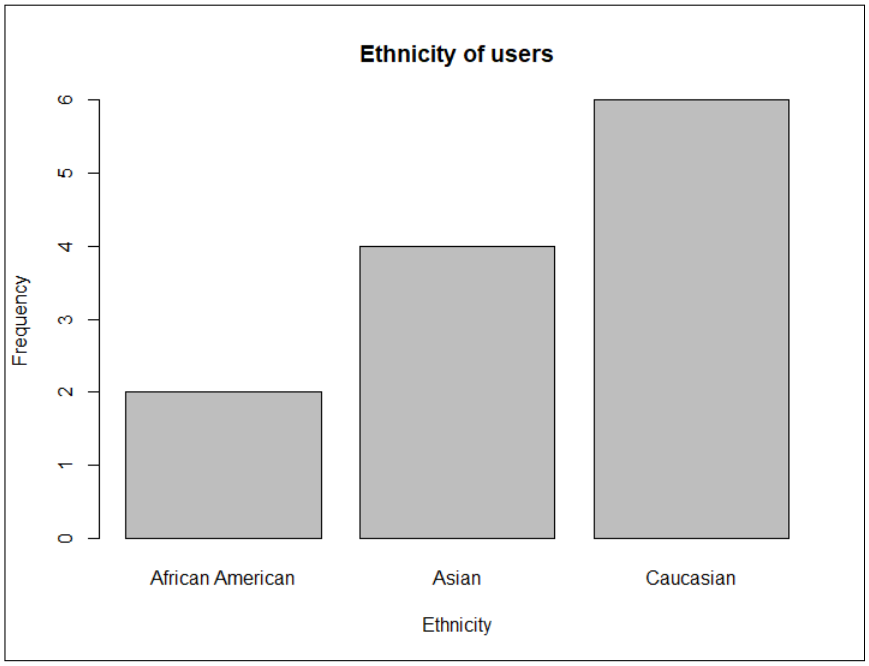
- There are many advanced methods to manage your dataset, you can find more in the appendix codes or in the reference book "R in Action".

1.4 Basic Plot

The following part will still focus on the credit_data.

- Bar Chart

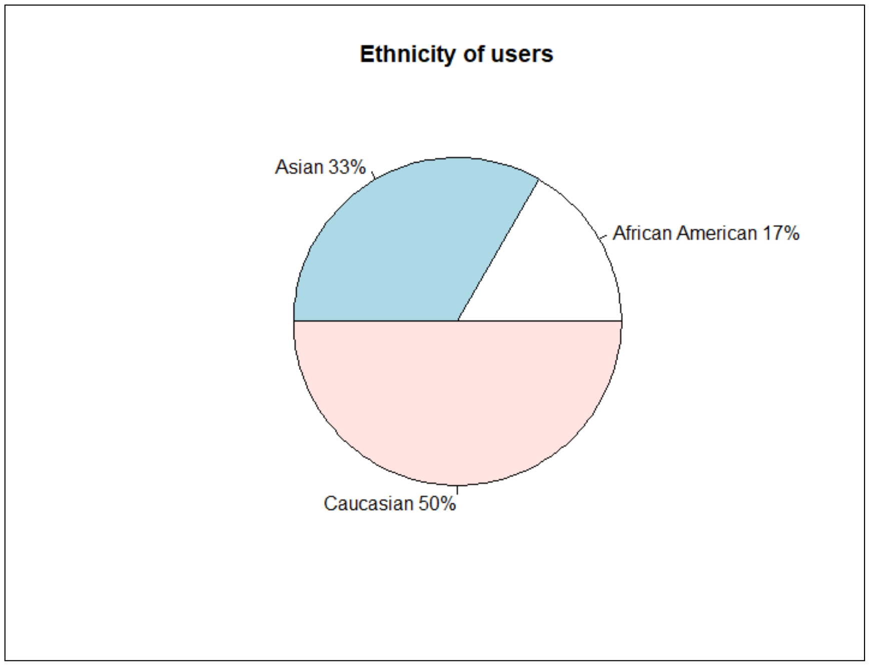
```
counts_1 <- table(credit_data$Ethnicity)
barplot(counts_1, main = "Ethnicity of users",
        xlab = "Ethnicity", ylab = "Frequency",
        horiz = FALSE)
```



- "main" is the title of the plot;
- "xlab" and "ylab" are the labels of the x-axis and y-axis;
- "horiz" indicates whether the bar chart is horizontal or not.

- Pie Chart

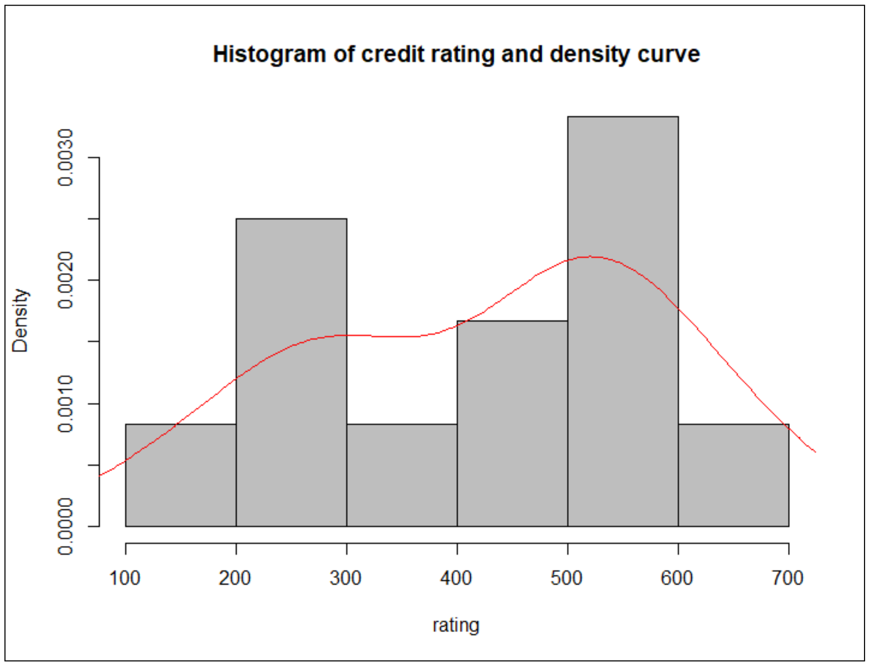
```
counts_1 <- table(credit_data$Ethnicity)
pct <- round(counts_1/sum(counts_1)*100)
lbls <- paste(names(counts_1), " ", pct, "%", sep = "")
pie(counts_1, labels = lbls,  main = "Ethnicity of users")
```



  - "round" function is used to round the number up/down;
  - "names" function extracts the column names of the table;
  - "paste" function combines different pieces of strings;
  - "labels" adds the labels to each fraction.

- Histogram

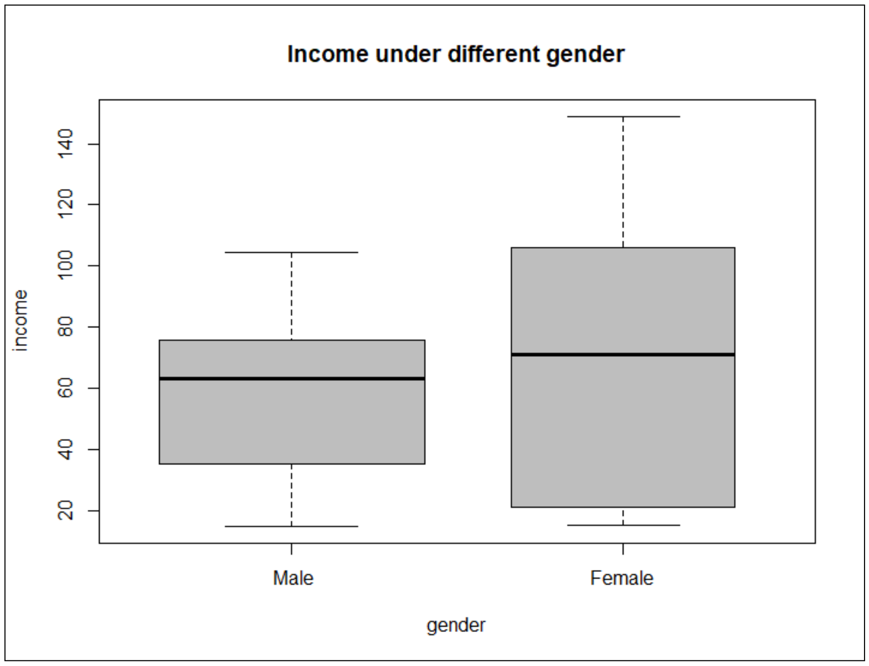
```
hist(credit_data$Rating,
     freq = FALSE, breaks = 6, col = "grey", xlab = "rating",
     main = "Histogram of credit rating and density curve")

lines(density(credit_data$Rating), col = "red")
```



- "freq" indicates whether the y-axis is frequency or density;
- "breaks" restricts the number of bins;
- "col" assigns the color of the histogram;
- "lines" and "density" functions add the density curve to the histogram.

- Box Plot

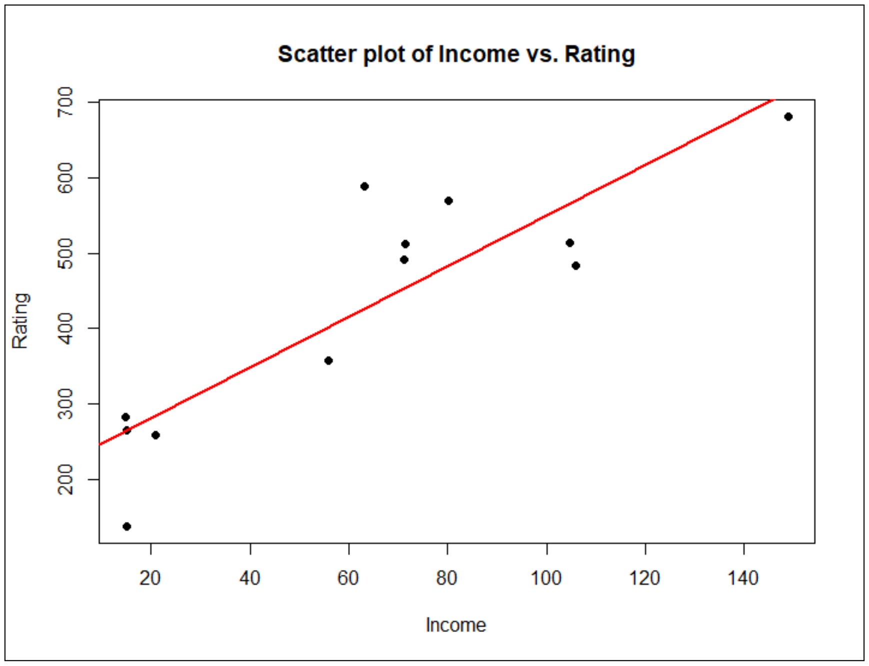
```
boxplot(Income ~ Gender, data = credit_data,
        notch  = FALSE, varwidth = TRUE, col = "grey",
        main = "Income under different gender",
        xlab = "gender", ylab = "income")
```



  - "Income ∼ Gender" gives the formula of the box plot;
  - "data" specifies what dataset you are using;
  - "notch" indicates whether the box plot is notched or not;
  - "varwidth" indicates whether box width is proportional to the sample size.

- Scatter Plot and Line Chart

```
attach(credit_data)
 plot(Income, Rating,
      main = "Scatter plot of Income vs. Rating",
      xlab = "Income", ylab = "Rating", pch = 19)
 abline(lm(Rating ~ Income), col = "red", lwd = 2, lty = 1)
detach(credit_data)
```



  - "attach()" and "detach()" let you focus on the dataset and avoid typing the name of the dataset every time (e.g. credit_data$Income);
  - "pch" specifies the shape of the points;
  - "abline" and "lm" functions add the linear fitting to the scatter plot;
  - "lwd" and "lty" indicate the width and the type of the fitting line.

1.5 Basic Statistical Analysis

- Sampling:

```r
set.seed(520)  # set seed for reproducibility

## Randomly select 3 observations without replacement
s0 <- sample(1:nrow(credit_data), 3, replace = FALSE)
credit_sample_test <- credit_data[s0, ]
credit_sample_train <- credit_data[-s0, ]
```

- Summary Statistics:

```r
myvars <- c("Income", "Limit", "Rating")
summary(credit_data[myvars])
```

```
     Income            Limit            Rating    
 Min.   : 14.89   Min.   :1311   Min.   :138.0  
 1st Qu.: 19.53   1st Qu.:3552   1st Qu.:278.8  
 Median : 67.08   Median :6732   Median :487.0  
 Mean   : 63.94   Mean   :5819   Mean   :428.5  
 3rd Qu.: 86.28   3rd Qu.:7347   3rd Qu.:527.8  
 Max.   :148.92   Max.   :9504   Max.   :681.0  
```

- Correlations:

```r
myvars <- c("Income", "Limit", "Rating")
cov(credit_data[myvars])                        # covariance
cor(credit_data[myvars], method = "pearson")    # pearson correlation
```

- t-test:

```r
t.test(Rating ~ Married, credit_data)
```

In this t-test, you are actually comparing the mean of credit ratings between married and unmarried people. In class question: what is the null hypothesis? If the p-value is 0.8271, what is your conclusion?

- Simple Linear Regression:
  - In this part, you will focus on the Boston housing price dataset. The population regression function is as follows (which means that the median of housing price is a linear function of the percentage of lower status of the population):

$$\text{medv} = \beta_0 + \beta_1 \times \text{lstat} + \epsilon$$

  - You want to use this linear model to predict the median of housing price. The first step is to split the dataset into training and testing sets and then fit the linear model using the training set.

```r
library(MASS)
library(ISLR)

nlines <- round(0.3 * nrow(Boston))  # 30% as test set
set.seed(520)
sampling <- sample(1:nrow(Boston), nlines, replace = FALSE)
Boston_test <- Boston[sampling, ]
Boston_train <- Boston[-sampling, ]

reg <- lm(medv ~ lstat, data = Boston_train)
summary(reg)  # estimation results
```

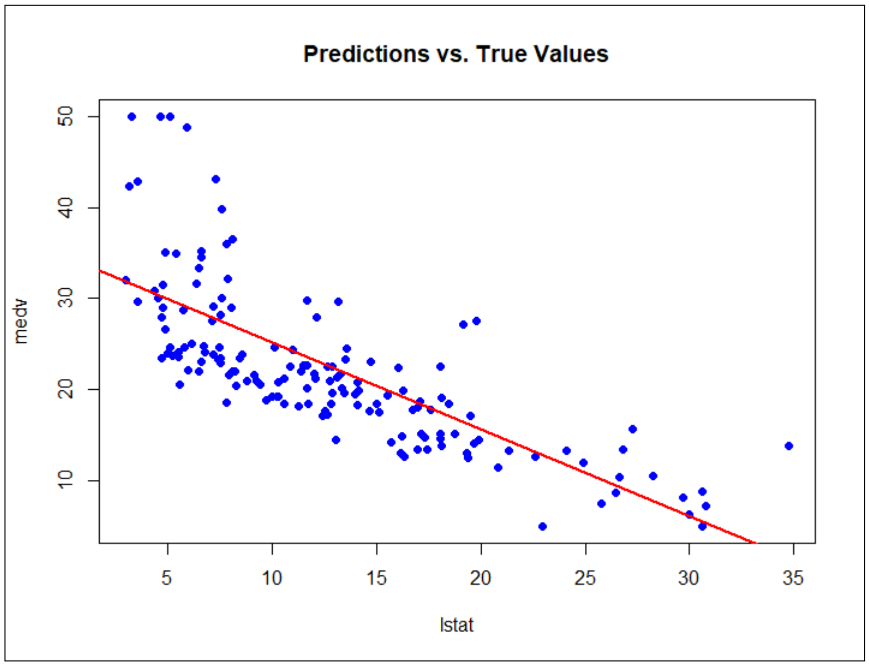
```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 34.77444    0.68877   50.49   <2e-16 ***
lstat       -0.95580    0.04737  -20.18   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

In class question:

1. What is the null hypothesis of this regression analysis?
2. Can you interpret the coefficient of “lstat”?
3. Are there any potential issues of this regression analysis?

- The second step is to use the fitted model to predict the median of housing price in the testing set and calculate the test error.

```r
pred <- predict(reg, newdata = Boston_test)
error <- mean((pred - Boston_test$medv)^2)

## Plot for Predictions and True Values
plot(Boston_test$lstat, Boston_test$medv,
     pch = 19, col = "blue",
     main = "Predictions vs. True Values",
     xlab = "lstat", ylab = "medv")
abline(reg, col = "red", lwd = 2, lty = 1)
```



- Please refer to the old version of tutorial slides for more details about statistical analysis, e.g. multiple linear regression.

Chapter 2

Basic Classification Methods

2.1 Data Description

- The dataset you are going to use is a stock data which has 1,250 observations and 9 variables. This dataset consists of percentage returns for the S&P 500 stock index over 1,250 days, from the beginning of 2001 until the end of 2005. The dependent variable, “Direction”, records whether the market is up or down on that day (which is a binary qualitative response variable). The goal is to use three features “Lag1”, “Lag2” and “Volume” to predict “Direction”.

- Before doing that, you can try some basic statistical analysis to get a sense of the data. For example, you can have the summary statistics and correlations. The following codes show you how to do that quickly (Results are not shown here).

```r
library(ISLR)

library(stargazer)
stargazer(Smarket[c("Lag1", "Lag2", "Volume")])

library(corrgram)
vars <- c("Lag1", "Lag2", "Volume")
corrgram(Smarket[vars], order = FALSE, lower.panel = panel.shade,
         upper.panel = panel.pie, text.panel = panel.txt,
         main = "Correlation of variables")

library(ggplot2)
Smarket$Direction <- as.factor(Smarket$Direction)
ggplot(data = Smarket, aes(x = Direction, y = Volume)) +
   geom_boxplot()
```

- You will use the data of 2001-2004 as training set and the data of 2005 of test set.

```r
attach(Smarket)
  train <- (Year < 2005)
  Stock_Data_test <- Smarket[!train, ]
  Stock_Data_train <- Smarket[train, ]
  Direction_2005 <- Direction[!train]
detach(Smarket)
```

2.2 KNN

- The first model you are going to use is KNN. Actually, you are using the following formula to estimate the conditional probability:

$$\Pr(Y = j|X = x) = \frac{1}{K}\sum_{i\in\mathcal{N}_0} I(y_i = j) \tag{2.1}$$

- You need to install the package “kknn”. Then, just as a linear regression, you can input the formula, training data, test data and the parameter k. Then you can check the classification result.

```r
install.packages("kknn")
library(kknn)
knn <- kknn(Direction ~ Lag1 + Lag2 + Volume,
            Stock_Data_train, Stock_Data_test, k = 3)
knn_pred <- fitted(knn)
table(Stock_Data_test$Direction, knn_pred, dnn = c("T", "P"))
```

- Confusion matrix:

	Prediction	
	Down	Up
Down	64	47
Up	79	62

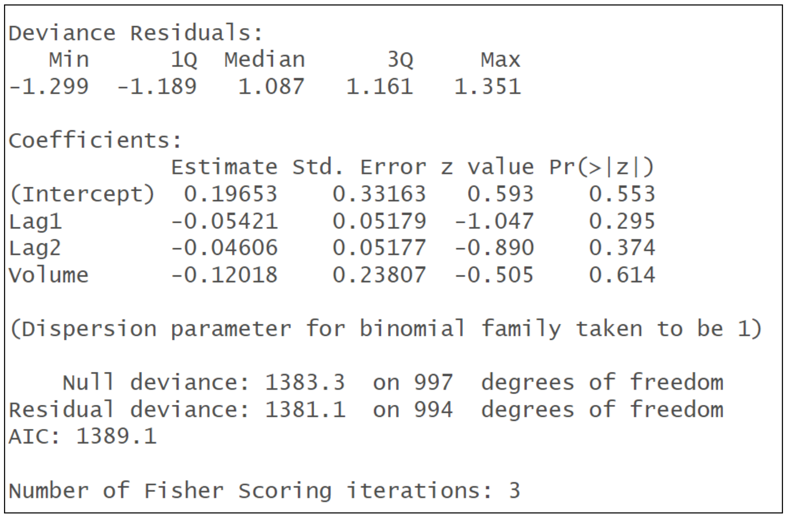
- In class question:
  1. Can you describe the procedure of KNN?
  2. How does the k affect the classification result in training set?
  3. How does the k affect the prediction result in test set?

2.3 Logit Regression

- The second method for classification is the Logistic regression.

```r
logit <- glm(Direction ~ Lag1 + Lag2  + Volume,
             data = Stock_Data, family = binomial(link = logit),
             subset = train)
summary(logit)
logit_probs <- predict(logit, Stock_Data_test, type = ”response”)
logit_pred <- rep(”Down”, 252)
logit_pred[logit_probs > 0.5] <- ”Up”
table(logit_pred, Direction_2005)
```

- Regression results:



```
Deviance Residuals:
   Min       1Q   Median       3Q      Max
-1.299   -1.189    1.087    1.161    1.351

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.19653    0.33163   0.593    0.553
Lag1        -0.05421    0.05179  -1.047    0.295
Lag2        -0.04606    0.05177  -0.890    0.374
Volume      -0.12018    0.23807  -0.505    0.614

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1383.3  on 997  degrees of freedom
Residual deviance: 1381.1  on 994  degrees of freedom
AIC: 1389.1

Number of Fisher Scoring iterations: 3
```

- Confusion matrix:

	Prediction	
	Down	Up
Down	79	32
Up	100	41

- In class question:
  1. You can try to replace the link function “logit” with “probit”. What is the similarity between these two link functions? How about the differences?
  2. What does logit_probs $>$ 0.5 mean? Can you change another number?
  3. How to estimate parameters β in logistic regression?

2.4 LDA and QDA

- Review the principle of LDA:
  - According to the Bayes theorem:

$$\begin{aligned}\Pr(Y=k|X=x) &= \frac{\Pr(Y=k)\cdot\Pr(X=x|Y=k)}{\Pr(X=x)} \\ &= \frac{\Pr(Y=k)\cdot\Pr(X=x|Y=k)}{\sum_{l=1}^{K}\Pr(Y=l)\cdot\Pr(X=x|Y=l)} \\ &= \frac{\pi_k f_k(x)}{\sum_{l=1}^{K}\pi_l f_l(x)}\end{aligned} \tag{2.2}$$

  - If you assume:

$$f_k(x)=\Pr(X=x|Y=k)=\frac{1}{\sqrt{2\pi}\sigma_k}\exp\left(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right) \tag{2.3}$$

  - Then you plug this conditional pdf back into the Bayes theorem and further assume that $\sigma_1^2=\sigma_2^2=...=\sigma_K^2=\sigma^2$:

$$\Pr(Y=k|X=x)=\frac{\pi_k\exp\left(-\frac{1}{2\sigma^2}(x-\mu_k)^2\right)}{\sum_{l=1}^{K}\pi_l\exp\left(-\frac{1}{2\sigma^2}(x-\mu_l)^2\right)} \tag{2.4}$$

  - Take log on both sides:

$$\begin{aligned}\log\Pr(Y=k|X=x) &= \log\pi_k-\frac{1}{2\sigma^2}(x-\mu_k)^2-B(x;\mu_l,\pi_l,\sigma) \\ &= \log\pi_k-\frac{1}{2\sigma^2}(x^2-2\mu_k x+\mu_k^2)-B(x;\mu_l,\pi_l,\sigma) \\ &= \log\pi_k-\frac{\mu_k^2}{2\sigma^2}+x\frac{\mu_k}{\sigma^2}-\left[\frac{x^2}{2\sigma^2}+B(x;\mu_l,\pi_l,\sigma)\right]\end{aligned} \tag{2.5}$$

  - Thus, you are comparing the following part:

$$\delta_k(x)=\frac{\mu_k}{\sigma^2}\cdot x+\left(\log\pi_k-\frac{\mu_k^2}{2\sigma^2}\right) \tag{2.6}$$

    which is a linear function of x.
  - In class question:
    1. How many parameters do you need to estimate for LDA in the above case? And how to estimate them?
    2. What if you have many predictors x? What is the assumption for LDA then?
    3. What is QDA? What is the difference between LDA and QDA?

- The third method is the LDA or QDA. First, you need to install “MASS”.

```r
library(MASS)
lda <- lda(Direction ~ Lag1 + Lag2  + Volume,
           data = Stock_Data, subset = train)
lda_pred <- predict(lda, Stock_Data_test)
table(lda_pred$class, Direction_2005)

qda <- qda(Direction ~ Lag1 + Lag2 + Volume,
           data = Stock_Data, subset = train)
qda_pred <- predict(qda, Stock_Data_test)
table(qda_pred$class, Direction_2005)
```

2.5 Naive Bayes

- The last method is the Naive Bayes. You need to install “klaR”.

```r
install.packages("klaR")
library(klaR)
Bayes1 <- NaiveBayes(Direction ~ Lag1 + Lag2 + Volume,
                     data = Stock_Data_train)
Bayes1[1:length(Bayes1)]
pre_Bayes1 <- predict(Bayes1, Stock_Data_test)
table(Stock_Data_test$Direction, pre_Bayes1$class)
```

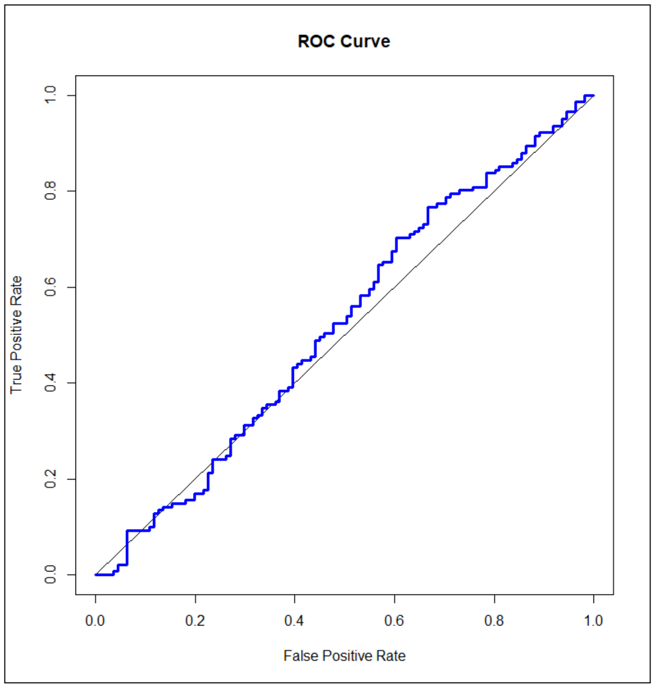
2.6 Comparison

1. KNN: $(64+62)/252 = 0.50$
2. Logit: $(79+41)/252 \approx 0.48$
3. LDA: $(79+41)/252 \approx 0.47$
4. QDA: $(84+31)/252 \approx 0.46$
5. Naive Bayes: $(84+31)/252 \approx 0.46$

2.7 ROC Curve

- You can manually draw the ROC Curve. Take LDA as an example.

```r
data <- cbind(Stock_Data_test, lda_pred$posterior)
threshold <- rep(NA, 10000)
FPR <- rep(NA, 10000)
TPR <- rep(NA, 10000)
attach(data)
  for (i in 0:10000){
     Down1 <- 1 * (Down > i / 10000)
     FPR[i] <- sum(Down1 == 0 & Direction == "Down") /
                sum(Direction == "Down")
     TPR[i] <- sum(Down1 == 0 & Direction == "Up") /
                sum(Direction == "Up")
  }
  xrange <- range(TPR)
  yrange <- range(TPR)
  plot(xrange, yrange, type = "l",
       xlab = "False Positive Rate", ylab = "True Positive Rate")
       lines(FPR, TPR, type = "s", col = "blue", lwd = 3)
  title("ROC Curve")
detach(data)
```



Chapter 3

Introduction to Model Selection

In this section, your main task is to use many features of a baseball player to predict his salary. The data contains 19 features and 263 observations. In the last part, you will learn a nonlinear method: cubic spline.

3.1 Subset Selection

- The first method is the subset selection method. You need to install "leaps". The following code block tells you how to run the best subset selection, forward selection and backward selection. (nvmax = 5 means that you are considering at most 5 predictors; If you do not announce the method, it will run the best subset selection.)

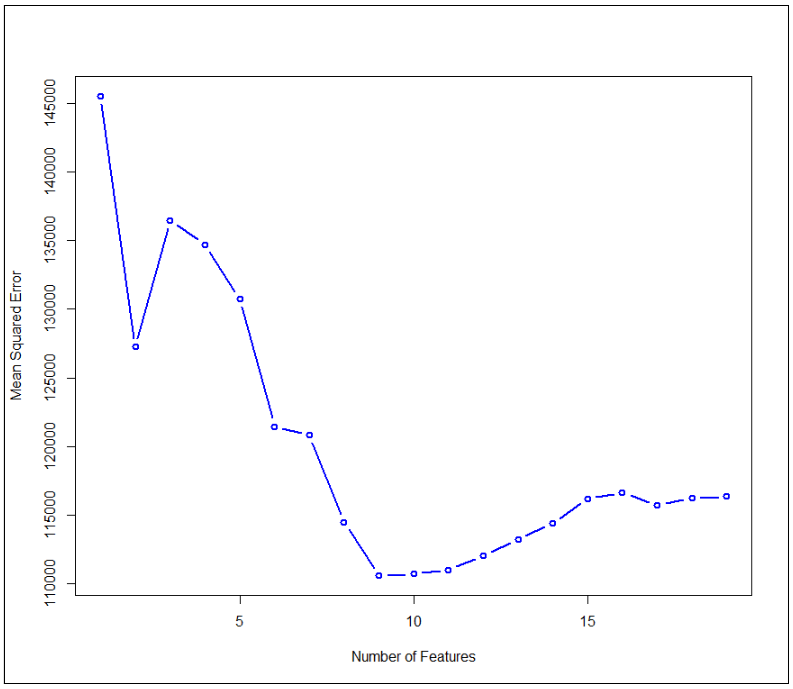
```r
library(ISLR)
Hitters <- na.omit(Hitters) # drop those observations with n.a.
install.packages("leaps")
library(leaps)

full <- regsubsets(Salary~., Hitters, nvmax=5)
fwd <- regsubsets(Salary~., Hitters, nvmax=5, method="forward")
bwd <- regsubsets(Salary~., Hitters, nvmax=5, method="backward")
```

- Unfortunately, the "leaps" package does not provide the function to do cross-validation. You need to write your own predict function to do that. For example,

```r
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
              }
```

- Then, you can do 10-fold cross-validation.

```r
k = 10 # number of folds
set.seed(123) # set the random seed
folds <- sample(1:k, nrow(Hitters), replace = TRUE)

# Generate a matrix to save the cv errors
cv.error <- matrix(NA, k, 19, dimnames = list(NULL, paste(1:19)))

for (j in 1:k) {
 bfit <- regsubsets(Salary~., data=Hitters[folds!=j,], nvmax=19)
 for (i in 1:19){
   pred <- predict.regsubsets(bfit, Hitters[folds==j,], id=i)
   cv.error[j,i] <- mean((Hitters$Salary[folds==j] - pred)^2)
 }
}

mvalue <- apply(cv.error, 2, mean)
plot(mvalue, type = "b", col = "blue", lwd = 2,
     xlab = "Number of Features",
     ylab = "Mean Squared Error")
```



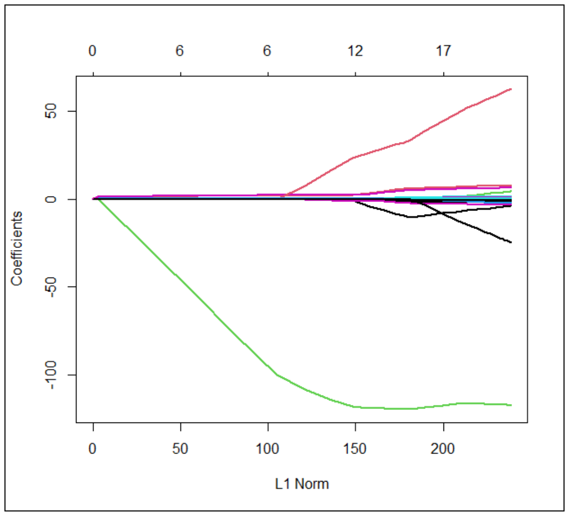
3.2 Shrinkage Methods

- The second method is the shrinkage method. You have learned ridge regression and lasso regression in the class. Now, there is a combination of these two, called elastic net method (See the following formula).

$$\min_{\beta} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{K} x_{i,j}\beta_j \right)^2 + \lambda \left[\alpha \|\beta\| + (1-\alpha) \|\beta\|^2 \right] \tag{3.1}$$

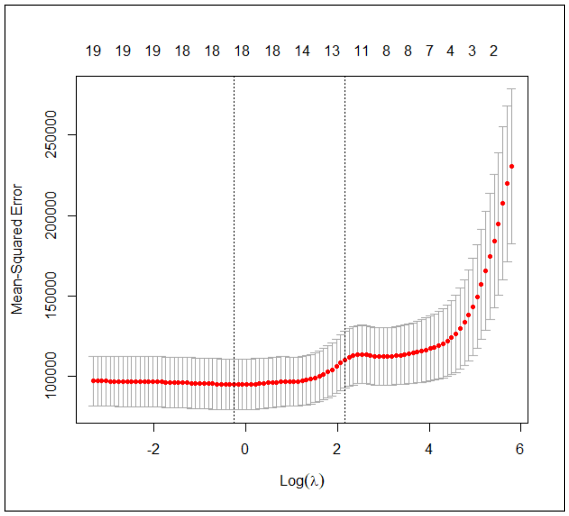
You can use the “glmnet” package to do that (also ridge and lasso).

```r
X <- model.matrix(Salary ~ ., Hitters)[,-1]
Y <- Hitters$Salary

install.packages("glmnet")
library(glmnet)

ridge.mod <- glmnet(X, Y, alpha = 0, lambda = 5)
lasso.mod <- glmnet(X, Y, alpha = 1, lambda = 5)
elasticnet.mod <- glmnet(X, Y, alpha = 0.5, lambda = 5)

# You can use the following code to plot the coefficients shrinkage
# Take lasso as an example:
grid <- 10^seq(10, -2, length = 100)
lasso.mod2 <- glmnet(X, Y, alpha = 1, lambda = grid)
plot(lasso.mod2, lwd = 2)
```



- In class question:
  1. Can ridge regression select variables? How about lasso and elastic net?
  2. Think about the unbiasedness and variance of the estimators in these three methods. Compare them with the OLS estimators.
- To select the best lambda, you can use cross-validation.

```r
set.seed(1997)
train <- sample(1:nrow(X), nrow(X)/2)
test <- (-train)
Y.test <- Y[test]

cv.out <- cv.glmnet(X[train,], Y[train], alpha = 1)
plot(cv.out)
bestlambda <- cv.out$lambda.min
bestlambda # 0.7712465
```



- In class question:
  1. Try “lambda.1se” rather than “lambda.min”. Why do people in industry prefer “lambda.1se”? Please provide some reasons.
  2. How can you draw the figure above without using the “cv.glmnet” function?
  3. In elastic net, how can you find the best combination of α and λ?
  4. Why do you scale (or standardize) the data before doing the shrinkage method?

3.3 Nonlinear Models

- Today, you will only focus on cubic spline. Other nonlinear models are quite similar. Firstly, review the theory part of cubic spline.

$$y_i = \begin{cases} \beta_{01} + \beta_{11}x_i + \beta_{21}x_i^2 + \beta_{31}x_i^3 + \epsilon_i & \text{if } x_i < c \\ \beta_{02} + \beta_{12}x_i + \beta_{22}x_i^2 + \beta_{32}x_i^3 + \epsilon_i & \text{if } x_i \geq c \end{cases} \tag{3.2}$$

  There are also three restrictions: (i) continuous at c; (ii) continuous first derivative at c; (iii) continuous second derivative at c which are equivalent to

$$\beta_{01} + \beta_{11}c + \beta_{21}c^2 + \beta_{31}c^3 = \beta_{02} + \beta_{12}c + \beta_{22}c^2 + \beta_{32}c^3 \tag{3.3}$$

$$\beta_{11} + 2\beta_{21}c + 3\beta_{31}c^2 = \beta_{12} + 2\beta_{22}c + 3\beta_{32}c^2 \tag{3.4}$$

$$2\beta_{21} + 6\beta_{31}c = 2\beta_{22} + 6\beta_{32}c \tag{3.5}$$

- The above equations can be written in one equation:

$$y_i = \alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 + \alpha_3 x_i^3 + \alpha_4 (x_i - c)_+^3 + \epsilon_i \tag{3.6}$$

  where $(x_i - c)_+^3 = (x_i - c)^3$ if $x_i > c$ and 0 otherwise.

- To show the equivalence:
  - If $x_i < c$, then: $y_i = \alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 + \alpha_3 x_i^3 + \epsilon_i$;
  - If $x_i > c$, then: $y_i = \alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 + \alpha_3 x_i^3 + \alpha_4(x_i^3 - 3cx_i^2 + 3c^2 x_i - c^3) + \epsilon_i$.
  - Rearrange the second equation:

$$y_i = \alpha_0 + (\alpha_1 + 3c^2\alpha_4)x_i + (\alpha_2 - 3c\alpha_4)x_i^2 + (\alpha_3 + \alpha_4)x_i^3 + \epsilon_i$$

  - At $x_i = c$, there must be:

$$\alpha_0 + \alpha_1 c + \alpha_2 c^2 + \alpha_3 c^3 = \alpha_0 + (\alpha_1 + 3c^2\alpha_4)c + (\alpha_2 - 3c\alpha_4)c^2 + (\alpha_3 + \alpha_4)c^3$$

$$\alpha_1 + 2\alpha_2 c + 3\alpha_3 c^2 = (\alpha_1 + 3c^2\alpha_4) + 2(\alpha_2 - 3c\alpha_4)c + 3(\alpha_3 + \alpha_4)c^2$$

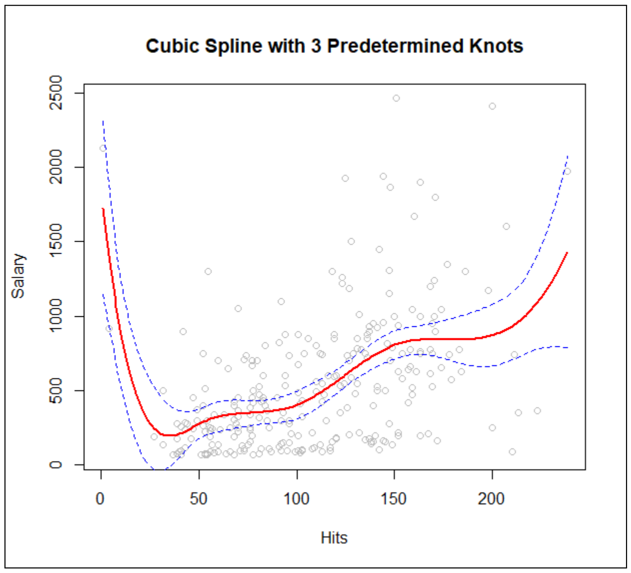
$$2\alpha_2 + 6\alpha_3 c = 2(\alpha_2 - 3c\alpha_4) + 6(\alpha_3 + \alpha_4)c$$

  - You will find that these three equations automatically hold.

- For the first version of expression of cubic spline, you need to use constrained optimization method to estimate the parameters. However, for the second version, you can just use OLS to estimate the parameters without any restrictions.

- In R, you can use the package “spline”.

```r
install.packages("splines")
library(splines)

lims <- range(Hitters$Hits)
grid <- seq(from = lims[1], to = lims[2])  # generate a grid

CSpline <- lm(Salary~bs(Hits, knots=c(50,100,150)), data=Hitters)
splinepred <- predict(CSpline, newdata=list(Hits=grid), se=T)

plot(Hitters$Hits, Hitters$Salary, col = "gray",
     xlab = "Hits", ylab = "Salary")
title("Cubic Spline with 3 Predetermined Knots")
lines(grid, splinepred$fit, lwd = 2, col = "red")
lines(grid, splinepred$fit + 1.96*splinepred$se,
      lty = "dashed", col = "blue")
lines(grid, splinepred$fit - 1.96*splinepred$se,
      lty = "dashed", col = "blue")
```



- In class question:
  1. How do you choose the knots? Can cross-validation work?
  2. If you want to use linear spline, how do you do it?

Chapter 4

Basic Tree-based Methods

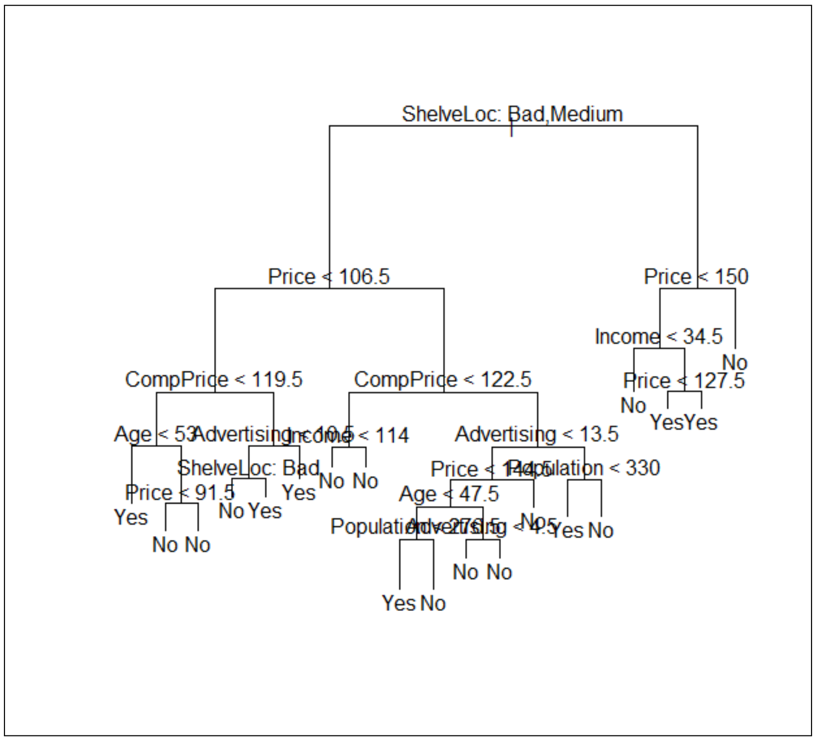
4.1 Classification Tree

- In this section, you will use a set of information to predict sales (high or low).
- Firstly, prepare the data and generate a test set and a training set.

```r
library(ISLR)
High <- as.factor(ifelse(Carseats$Sales <= 8, "No", "Yes"))
Data <- data.frame(Carseats, High)

set.seed(911)
train <- sample(1:nrow(Data), 200)
Testset <- Data[-train, ]
High.test <- High[-train]
```

- Then, you can use the “tree” package to generate a classification tree.

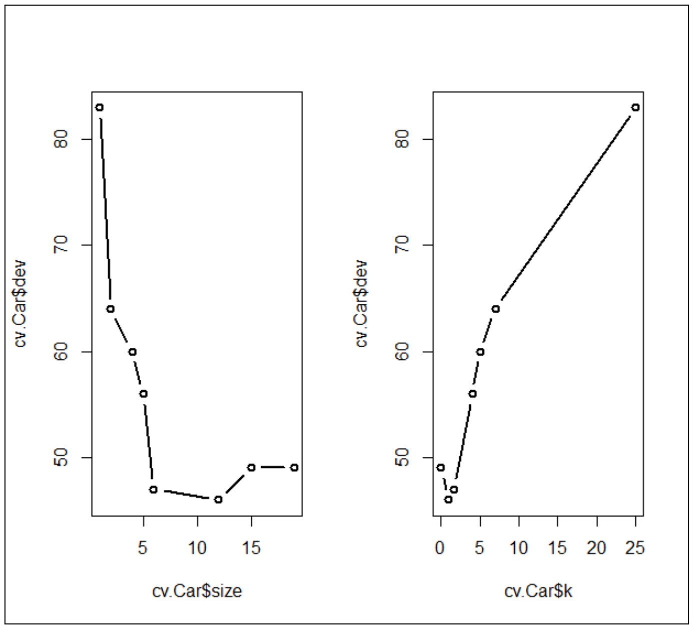
```r
install.packages("tree")
library(tree)
TrainTree <- tree(High ~ . -Sales, Data, subset = train)
plot(TrainTree)
text(TrainTree, pretty = 0)

Pred1 <- predict(TrainTree, Testset, type = "class")
table(Pred1, High.test)
```

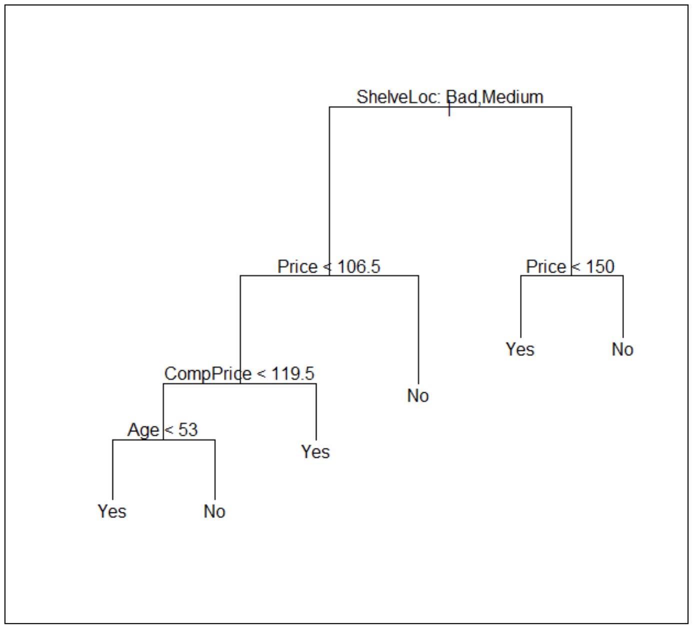


- The confusion matrix tells us, the accuracy of prediction is about 68%. Then, you can try to prune the tree to improve the accuracy.

```r
cv.Car <- cv.tree(TrainTree, FUN = prune.misclass)
par(mfrow = c(1, 2))
plot(cv.Car$size, cv.Car$dev, type = "b", lwd = 2)
plot(cv.Car$k, cv.Car$dev, type = "b", lwd = 2)

prune.Car <- prune.misclass(TrainTree, best = 6)
par(mfrow = c(1, 1))
plot(prune.Car)
text(prune.Car, pretty = 0)

Pred2 <- predict(prune.Car, Testset, type = "class")
table(Pred2, High.test)
```

- From the plot below, a tree with 6 terminal nodes may be a good choice (for some reasons). Then, the new tree is like the following figure. The accuracy of prediction is about 69.5% which is greater than before. In class question: why?

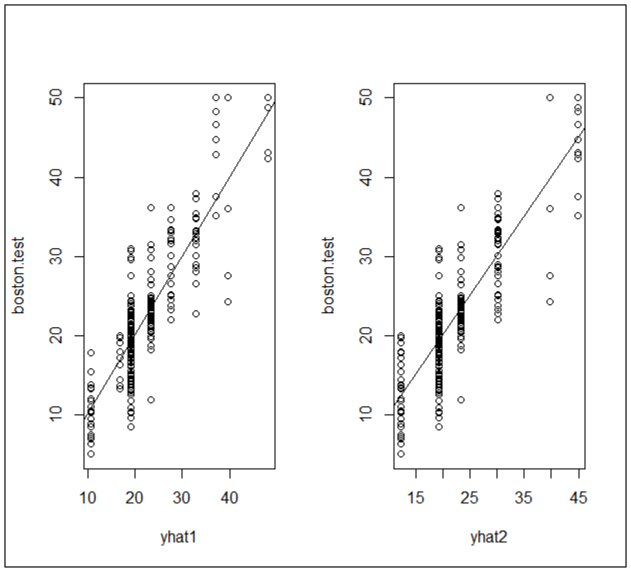

cv.Car$dev
cv.Car$size
cv.Car$k
50
60
70
80
5
10
15
0
20
25


ShelveLoc: Bad,Medium
Price < 106.5
Price < 150
CompPrice < 119.5
Age < 53
Yes
No
Yes
No
Yes
No

4.2 Regression Tree

- Similarly, you can run a regression tree. Revisit the Boston data.

```r
library(MASS)
set.seed(911)
train <- sample(1:nrow(Boston), nrow(Boston)/2)

library(tree)
tree.boston <- tree(medv ~ ., data = Boston, subset = train)
cv.boston <- cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type = "b")
prune.boston <- prune.tree(tree.boston, best = 6)

yhat1 <- predict(tree.boston, newdata = Boston[-train, ])
yhat2 <- predict(prune.boston, newdata = Boston[-train, ])
boston.test <- Boston[-train, "medv"]

par(mfrow = c(1, 2))
plot(yhat1, boston.test)
abline(0, 1)
plot(yhat2, boston.test)
abline(0, 1)

mean((yhat1 - boston.test)^2) # 20.14976
mean((yhat2 - boston.test)^2) # 19.75798
```



4.3 Bagging and Random Forest

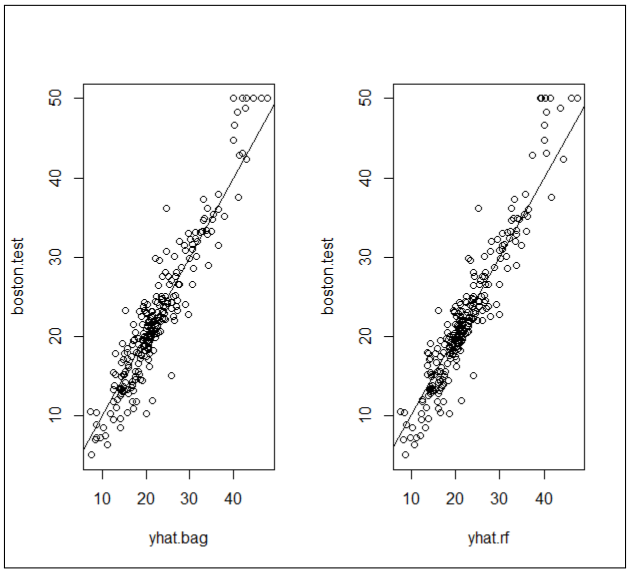
- In this part, you will learn two methods, bagging and random forest to further improve the prediction accuracy. The code is as follows. You need to install "randomForest".

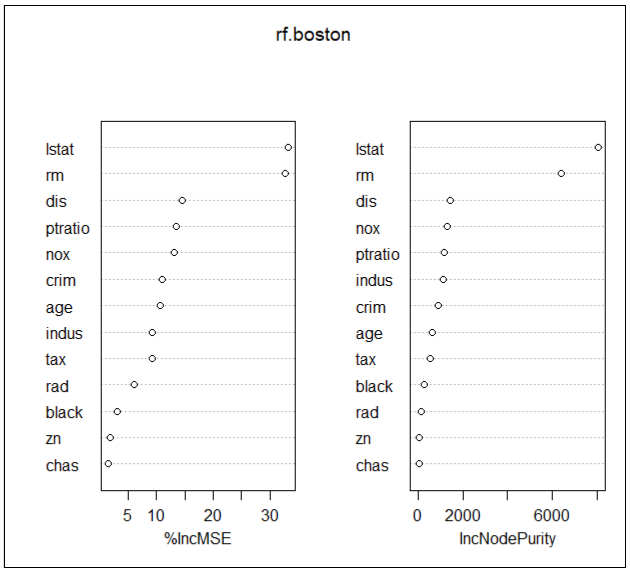
```r
install.packages("randomForest")
library(randomForest)
set.seed(911)
bag.boston <- randomForest(medv ~ ., data = Boston, subset = train,
                 mtry = 13, importance = TRUE, ntree = 25)
####  mtry = the number of variables, then randomforest == bagging
bag.boston

yhat.bag <- predict(bag.boston, newdata = Boston[-train, ])
plot(yhat.bag, boston.test)
abline(0, 1)
mean((yhat.bag - boston.test)^2) # 9.348806

####  randomForest
set.seed(911)
rf.boston <- randomForest(medv ~., data = Boston, subset = train,
                           mtry = 6, importance = TRUE)
yhat.rf <- predict(rf.boston, newdata = Boston[-train, ])
plot(yhat.rf, boston.test)
abline(0, 1)
mean((yhat.rf - boston.test)^2) # 9.130221

importance(rf.boston)
varImpPlot(rf.boston)
```

  - "mtry" is the number of variables randomly selected as candidates at each split.
  - "ntree" is the number of trees to grow.
  - "importance" is to calculate the importance of each variable.

- In class question:

  1. Why can you set random seeds in the above code?
  2. Why do bagging and random forest outperform a single tree?
  3. Why does the following plot (the results of prediction of bagging and random forest) look like this? What is special when comparing with the same plot of a single tree (or a pruned single tree)?


boston.test
yhat.bag
boston.test
yhat.rf

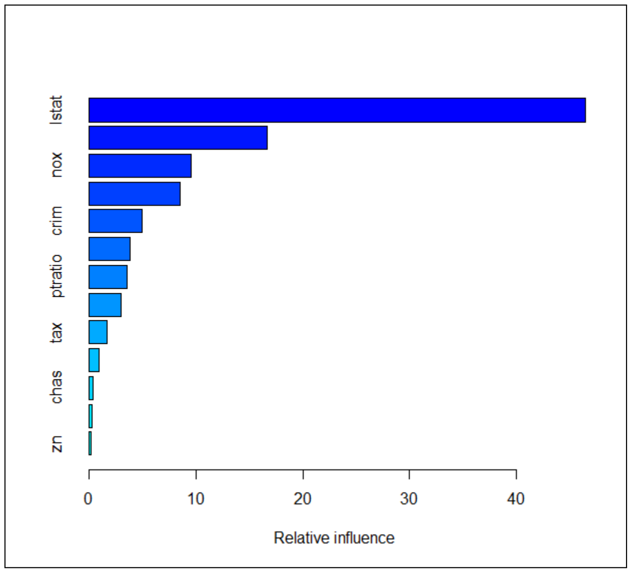

rf.boston
lstat
rm
dis
ptratio
nox
crim
age
indus
tax
rad
black
zn
chas
%IncMSE
lstat
rm
dis
nox
ptratio
indus
crim
age
tax
black
rad
zn
chas
IncNodePurity

4.4 Boosting

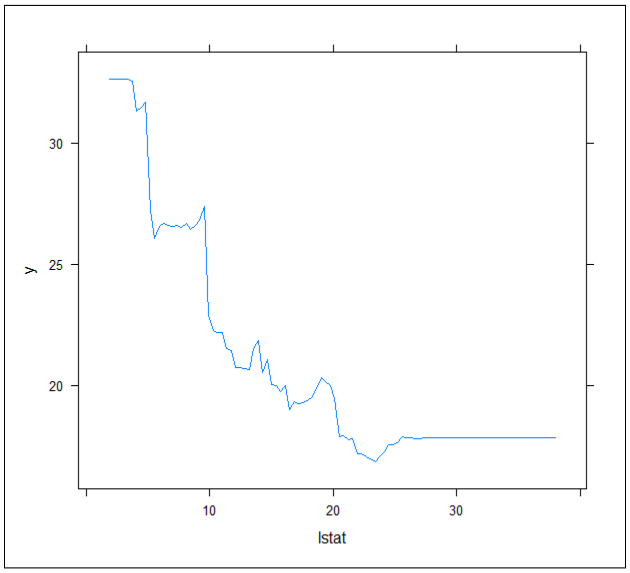
- The final method is boosting. You need to install “gbm”.

```r
install.packages("gbm")
library(gbm)

set.seed(911)
boost.boston <- gbm(medv ~., data = Boston[train, ],
                    distribution = "gaussian",
                    n.trees = 5000, interaction.depth = 4,
                    shrinkage = 0.2)
summary(boost.boston)
plot(boost.boston, i = "lstat")
yhat.boost <- predict(boost.boston, newdata = Boston[-train, ],
                      n.trees = 5000)
mean((yhat.boost - boston.test)^2) # 11.52177
```

  - “distribution” is the distribution of the response variable.
  - “n.trees” is the number of trees to grow.
  - “interaction.depth” is the maximum depth of each tree.
  - “shrinkage” is the parameter λ that controls for the learning speed.



- There are a lot of boosting algorithms, e.g. AdaBoost, XgBoost. If you are interested in those extensions, please refer to more advanced textbooks or online materials.



4.5 More Discussions

- In this example, random forest provides the most accurate prediction in the test set. Then, bagging and boosting. A single tree and its pruned version are not good.
- Think about the difference between regression tree and regression with dummies? What is the main difference between these two methods?
- What are advantages of tree methods comparing with others you learned before?

Chapter 5

Introduction to Deep Learning

5.1 Single-layer Neural Network

- You will still use the Hitters data to build up a single-layer neural network. First, you need to install "reticulate", "keras" and "tensorflow" to set up the environment.

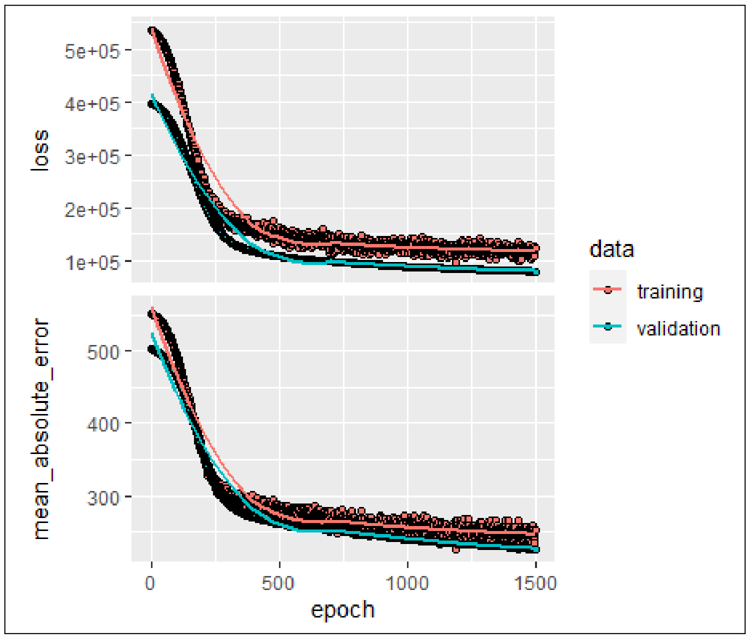
```r
install.packages("reticulate")
install.packages("keras")
install.packages("tensorflow")
library(keras)
install_keras()
```

- Then, you split the data into training set and test set.

```r
library(ISLR2)
Gitters <- na.omit(Hitters)
set.seed (911)
ntest <- trunc(nrow(Gitters) / 3)
testid <- sample (1:nrow(Gitters), ntest)

x <- scale(model.matrix(Salary ~ . - 1, data = Gitters))
y <- Gitters$Salary
```

- Next, you can use keras to build up a single-layer neural network. The pipe operator %>% passes the previous term as the first argument to the next function, and returns the result. It makes the codes more readable.

```r
library(keras)
modnn <- keras_model_sequential() %>%
 layer_dense(units = 50, # single layer with 50 units
 activation = "relu", # a ReLU activation function
 input_shape = ncol(x)) %>%
 layer_dropout(rate = 0.4) %>% # dropout rate 0.4
 layer_dense(units = 1)  # single output unit
## "input_shape" is required when building up the 1st layer.

## Add details to modnn that control the fitting algorithm
modnn %>% compile(loss = "mse",
          optimizer = optimizer_rmsprop(),
          metrics = list("mean_absolute_error"))
## Fit the model by using the training data
history <- modnn %>% fit(x[-testid , ], y[-testid],
           epochs = 1500, batch_size = 32,
           validation_data = list(x[testid , ], y[testid]))

library(ggplot2)
plot(history)

## Predict from the final model
npred <- predict(modnn, x[testid , ])
mean(abs(y[testid] - npred)) # 226.3954
```



- You can compare the prediction accuracy with other methods.

```
## Linear Regression
lfit <- lm(Salary ~ ., data = Gitters[-testid, ])
lpred <- predict(lfit, Gitters[testid, ])
with(Gitters[testid, ], mean(abs(lpred - Salary))) # 210.5706

## Lasso Regression
library(glmnet)
cvfit <- cv.glmnet(x[-testid, ], y[-testid], type.measure = "mae")
cpred <- predict(cvfit, x[testid, ], s = "lambda.min")
mean(abs(y[testid] - cpred)) # 192.1307
```

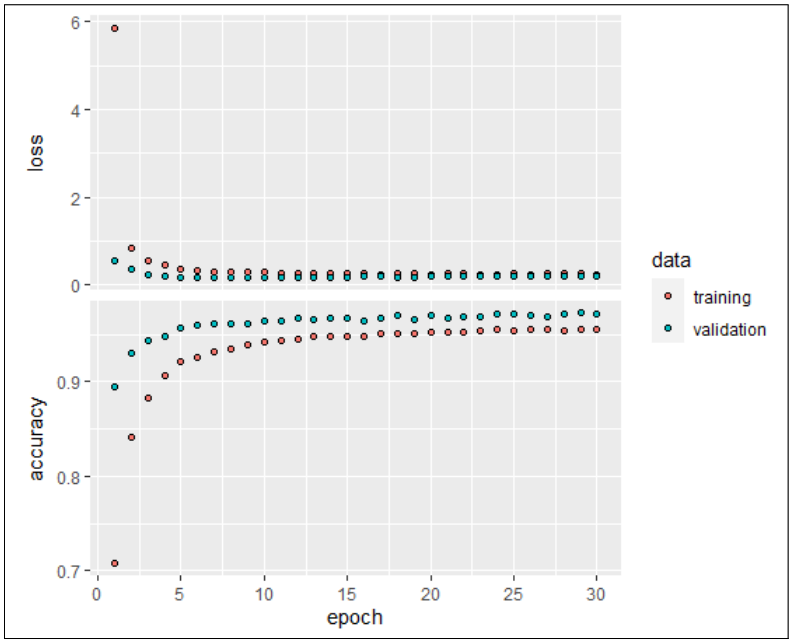
5.2 Multi-layer Neural Network

- There are 60,000 images in the training data and 10,000 in the test data. The images are 28×28, and stored as a three-dimensional array, so we need to reshape them into a matrix. Also, we need to "one-hot" encode the class label. Luckily keras has a lot of built-in functions that do this for us.

```
## Import the data (3-D array)
library(keras)
mnist <- dataset_mnist()
x_train <- mnist$train$x
g_train <- mnist$train$y
x_test <- mnist$test$x
g_test <- mnist$test$y
dim(x_train) # 60000 28 28
dim(x_test) # 10000 28 28

## Reshape them into a matrix
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
y_train <- to_categorical(g_train, 10)
y_test <- to_categorical(g_test, 10)
dim(x_train) # 60000 784
dim(x_test) # 10000 784

## Re-scale to the unit interval (0~255)
x_train <- x_train / 255
x_test <- x_test / 255
```

- Then, you can build up a multi-layer neural network.

```r
##  Set up the model with 3 layers
modelnn <- keras_model_sequential ()
modelnn %>%
  layer_dense(units = 256, activation = "relu",
              input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = "softmax")

summary(modelnn)

##  Define the loss function
modelnn %>% compile(loss = "categorical_crossentropy",
   optimizer = optimizer_rmsprop (),
   metrics = c("accuracy"))

system.time(  ##  record the time
   history <- modelnn %>%
   fit(x_train , y_train , epochs = 30, batch_size = 128,
   validation_split = 0.2)) ## 20% of the data are used to test
plot(history, smooth = FALSE)
```



- Define a function that helps you make predictions and test the accuracy.

```
## "predict_classes()" was removed in Tensorflow 2.6
accuracy <- function(pred , truth) {
          mean(drop(pred) == drop(truth))
        }

## modelnn %>% predict_classes(x_test) %>% accuracy(g_test)
## instead, you should use the following codes
y_prob <- modelnn %>% predict(x_test)
y_predict <- apply(y_prob, 1, which.max) - 1
accuracy(y_predict, g_test) # 0.9724
```

- Compare with multi-class logistic regression.

```
modellr <- keras_model_sequential () %>%
layer_dense(input_shape = 784, units = 10, activation = "softmax")
summary(modellr)

modellr %>% compile(loss = "categorical_crossentropy",
   optimizer = optimizer_rmsprop (), metrics = c("accuracy"))

modellr %>% fit(x_train , y_train , epochs = 30,
                batch_size = 128, validation_split = 0.2)

y_prob2 <- modellr %>% predict(x_test)
y_predict2 <- apply(y_prob2, 1, which.max) - 1
accuracy(y_predict2, g_test) # 0.9008
```

5.3 Convolutional Neural Network

- In this section you will fit a CNN to the CIFAR100 data, which is available in the keras package. It is arranged in a similar fashion as the MNIST data. The array of 50,000 training images has four dimensions: each three-color image is represented as a set of three channels, each of which consists of 32×32 eight-bit pixels. You standardize as you did for the digits, but keep the array structure. Then, you one-hot encode the response factors to produce a 100-column binary matrix.

```
## Import data
cifar100 <- dataset_cifar100 ()
names(cifar100)

x_train <- cifar100$train$x
g_train <- cifar100$train$y
x_test <- cifar100$test$x
g_test <- cifar100$test$y
dim(x_train)
range(x_train[1,,, 1])

## Standardize the data
x_train <- x_train / 255
x_test <- x_test / 255
y_train <- to_categorical(g_train , 100)
dim(y_train)

## Build up CNN model
model <- keras_model_sequential () %>%
   layer_conv_2d(filters = 32, kernel_size = c(3, 3),
                 padding = "same", activation = "relu",
                 input_shape = c(32, 32, 3)) %>%
   layer_max_pooling_2d(pool_size = c(2, 2)) %>%
   layer_conv_2d(filters = 64, kernel_size = c(3, 3),
                 padding = "same", activation = "relu") %>%
   layer_max_pooling_2d(pool_size = c(2, 2)) %>%
   layer_conv_2d(filters = 128, kernel_size = c(3, 3),
                 padding = "same", activation = "relu") %>%
   layer_max_pooling_2d(pool_size = c(2, 2)) %>%
   layer_conv_2d(filters = 256, kernel_size = c(3, 3),
                 padding = "same", activation = "relu") %>%
   layer_max_pooling_2d(pool_size = c(2, 2)) %>%
   layer_flatten () %>%
   layer_dropout(rate = 0.5) %>%
   layer_dense(units = 512, activation = "relu") %>%
   layer_dense(units = 100, activation = "softmax")

model %>% compile(loss = "categorical_crossentropy",
                  optimizer = optimizer_rmsprop(),
                  metrics = c("accuracy"))

history <- model %>% fit(x_train , y_train , epochs = 30,
                         batch_size = 128, validation_split = 0.2)

y_prob3 <- model %>% predict(x_test)
y_predict3 <- apply(y_prob3, 1, which.max) - 1
accuracy(y_predict3 , g_test)
```