STATS 415 Labs Summary

Chongxing Fan (University of Michigan, Ann Arbor)

April 17, 2020

1 Basics

1.1 Operators

Variable Assignment	
x <- 3	Variable Assignment
x = 3	Variable Assignment (not recommended)
Arithmetic	
x + y	Addition
x - y	Subtraction
x * y	Numerical Product
x %*% y	Matrix Multiplication
x ^ y	Power
x %% y	Mod
x / y	Division
Logical	
x < 3	Smaller than
x > 3	Greater than
x <= 3	Smaller than and equal to
x >= 3	Greater than and equal to
x == 3	Equal to
x != 3	Not equal to
a %in% b	If elements in a is also in b
a & b	Logical AND
a | b	Logical OR
!a	Logical NOT
Miscellaneous	
D$x	Column Selector
y ~ x	Depend on

1.2 Variables in Workspace

ls()	List all of the objects created
print(var)	Print out the variable *var*
typeof(var)	The type of value stored by *var*
as.factor(var)	Convert the variable to type *factor*
as.numeric(var)	Convert the variable to type *numeric*
as.character(var)	Convert the variable to type *character*

1.3 Constants and Useful Functions

NULL	Null value
sqrt(x)	Square root
floor(x)	Round down to an integer
ceiling(x)	Round up to an integer
round(x, n)	Round to the number with n digits
set.seed(n)	Set seed for random number generator
rnorm(n)	Generate n random numbers following a standard normal distribution
rnorm(n, mean=m, sd=s)	Generate n random numbers following a given normal distribution
sample(1:10, size=3)	Random sampling within a variable
order(v, decreasing)	The index of elements in a sorted way

1.4 Package Management

install.packages('PACKAGE')	Install package *"PACKAGE"*
library(PACKAGE)	Load package to the workspace

1.5 Flow Control

Conditional

```
if (condition) {
    do sth
} else {
    do sth
}
```

For Loop

```
for (i in start:end) {
    do sth
}
```

1.6 Vectors and Matrices

Creation	
V <- c(1, 2, 3)	Create a vector containing values 1, 2, 3
V <- 1:5	A vector ranging from 1 to 5
seq(start, end, step)	Create an even-step sequence
matrix(1:25, nrow=5, ncol=5)	Create a matrix
diag(1:5)	Diagonal matrices
Indexing	
V[1:2]	The first two elements of V
V[-2]	All elements except the second
V[c(1,3)]	The first and the third elements
V[length(x)]	The last element
V[x<3]	Conditional indexing
Inquiry	
length(V)	The size of the vector V
which(V<3)	Returns the indices of the TRUE values
dim(A)	Dimension of a matrix

nrow(A)	Row count of a matrix
ncol(A)	Column count of a matrix
Manipulation	
rep(V, times=n)	Repeat V by n times
rep(V, each=n)	Repeat each element in V by n times
t(A)	Matrix Transpose
rbind(A, B)	Matrix concatenation row-wise
cbind(A, B)	Matrix concatenation column-wise
sort(V, decreasing=TRUE)	Sort the array

* Arithmetic operators are applied element by element

1.7 Data Frames

Inquiry	
str(D)	Structure of the data frame
head(D)	First few rows of the data frame
names(D)	Column names of a data frame
summary(D)	Quantitative summaries
levels(D$x)	The levels of a categorical variable
nlevels(D$x)	The number of levels of a categorical variable
table(D$x)	Count the number of cases in each category
unique(D$x)	Unique values of the variable
Manipulation	
D$x2 <- x2_var	Add variables
subset(D, x1>=4)	Filter the data frame by conditions
tapply(Dy, Dx, func)	Apply the function to y within the category x
aggregate(y x, FUN=mean, data=D)	Apply the function to y within the category x and summarize
Indexing	
D[x<4.7, c('VarName1','VarName2')]	Indexing a data frame by rows and columns

1.8 Statistics & Visualization

max(x)	Max number
min(x)	Min number
mean(x)	Mean value
var(x)	Variance
sd(x)	Standard deviation
corr(D)	Correlation
scale(D$x, center=m, scale=k)	Standardize the variables
hist(D$x)	Histograms
boxplot(D$x)	Boxplots
abline(a,b)	Line y=a+bx
legend(loc, legend, col, cex, lwd, lty)	Add a legend
plot(as.factor(D$x))	Bar chart for categorical variables
plot(x,y)	Scatterplot
pairs(D)	Pairwise scatterplot matrix

1.9 File I/O

getwd()	Get Working Directory
read.table('a.csv', header=TRUE, sep=',')	Read an ASCII table with given separator

1.10 RMarkdown

Blank line	Paragraph break (hard break)
Two blank spaces at the end of a line	Line break (soft break)
# Header	Headers (the number of #s indicates the level)
Word	Italicize
Word	Bold
* Item	Bullet Lists
- Item	Bullet Lists
1. Item	Numbered Lists
``r	Code Chunks
`r arg`	Inline Code
$$x=y$$	Math Chunks
$x=y$	Inline Math

2 Linear Regression

2.1 Model

```
mod1 <- lm(y ~ x, data=D)
```

Use 'summary' to show the estimates

```
## Call:
## lm(formula = y ~ x, data = D)
##
## 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 ***
## x           -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
```

2.2 Variables and Functions

mod1$coefficients	Coefficients (including intercept)
mod1$residuals	Residuals
mod1$fitted.values	Fitted values
coef(mod1)	Coefficients (including intercept)
residuals(mod1)	Residuals

fitted(mod1)	Fitted values
predict(mod1, newdata)	Predict the values using the model given a new data frame *newdata*

2.3 Residual Plots

```
plot(mod1$residuals ~ mod1$fitted.values, main="TITLE",
     xlab="Fitted values", ylab="Residuals")
abline(a=0, b=0, col="gray60")
```

2.4 Quadratic Term

```
mod1 <- lm(y ~ x + I(x^2), data=D)
```

2.5 Multivariate Linear Regression

```
mod1 <- lm(y ~ x1 + x2 + x3 + x4, data=D)
```

2.6 Interaction

```
mod1 <- lm(y ~ x1 + x2 + x1:x2, data=D)
mod2 <- lm(y ~ x1*x2, data=D)
```

Here, mod1 and mod2 are equivalent.
Main effects must be included if the interaction term is present.

2.7 Categorical Variables

Categorical variable is accepted and no need to code it to numeric values.
To change the baseline level, use *factor*

```
D$x1 <- factor(D$x1, levels=c("Lvl2", "Lvl1", "Lvl3"))
```

2.8 ANOVA Table and F Test

To test whether a new model has a significant improvement compared with the original one, we use the ANOVA table and the F-statistics.

```
anova(mod1, mod2)
```

The result is given below.

```
## Analysis of Variance Table
##
## Model 1: medv2019 ~ lstat
## Model 2: medv2019 ~ lstat + rm + chas + lstat:chas
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    504 897030
## 2    501 682582  3    214448 52.467 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If the new model is significant (***), the new model has a significant improvement against the original one. (in this case, the three additional predictors are useful.)

3 K Nearest Neighbors (KNN)

3.1 Regression Model

```
install.packages("FNN")
library(FNN)
knnTrain <- knn.reg(train = trainData[,"x"], y = trainData$y,
                    test = trainData[,"x"], k = 5)
trainMSE <- mean((trainData$y - knnTrain$pred)^2)
knnTest <- knn.reg(train = trainData[,"x"], y = trainData$y,
                   test = testData[,"x"], k = 5)
testMSE <- mean((testData$y - knnTest$pred)^2)
```

3.2 Classification Model

```
install.packages("FNN")
library(FNN)
knnTrain <- knn(train = trainData[,"x"], cl = trainData$y,
                    test = trainData[,"x"], k = 5)
trainErr <- mean(knnTrain != trainData$y)
knnTest <- knn(train = trainData[,"x"], cl = trainData$y,
                test = testData[,"x"], k = 5)
testErr <- mean(knnTest != testData$y)
```

3.3 Split the training and test samples

```
set.seed(1997)
train_size <- floor(nrow(Boston) * 0.7)
train_id <- sample(1:nrow(Boston), train_size)
trainBoston <- Boston[train_id, ]
testBoston <- Boston[-train_id, ]
```

3.4 MSE-$1/K$ Plot

```
vars <- c("x1","x2")
k_range <- c(1,2,5,10,20,50,100)
trainMSE <- c()
testMSE <- c()

for (i in 1:length(k_range)) {
    knnTrain <- knn.reg(train = trainData[,vars],
                        y = trainData$y,
                        test = trainData[,vars],
                        k = k_range[i])
    trainMSE[i] <- mean((trainData$y - knnTrain$pred)^2)
    knnTest <- knn.reg(train = trainData[,vars],
                       y = trainData$y,
                       test = testData[,vars],
                       k = k_range[i])
    testMSE[i] <- mean((testData$y - knnTest$pred)^2)

plot(trainMSE ~ I(1/k_range), type="b", lwd=2, col="blue",
     main="TITLE", xlab="1/K", ylab="MSE")
lines(testMSE ~ I(1/k_range), type="b", lwd=2, col="red")
legend("topright", legend=c("Training KNN", "Test KNN"),
       cex=0.75, col=c("blue", "red"), lwd=c(2,2),
       pch=c(1,1), lty=c(1,1))
```

4 LDA & QDA

4.1 Linear Discriminant Analysis (LDA)

```r
install.packages("MASS")
library(MASS)
lda_model <- lda(y ~ x1 + x2, data = D)
lda_model
```

The example output is:

```
## Call:
## lda(Species ~ Sepal.Width + Sepal.Length, data = iris_train)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Width Sepal.Length
## setosa        3.448571     4.960000
## versicolor    2.831429     6.011429
## virginica     2.977143     6.522857
##
## Coefficients of linear discriminants:
##                     LD1        LD2
## Sepal.Width   2.947224 -2.2629106
## Sepal.Length -2.162274 -0.8742136
##
## Proportion of trace:
##    LD1    LD2
## 0.9775 0.0225
```

4.2 Quadratic Discriminant Analysis (QDA)

```r
install.packages("MASS")
library(MASS)
qda_model <- qda(y ~ x1 + x2, data = D)
qda_model
```

4.3 Variables and Functions

pred <- predict(mod, data)	Predict using the model
pred$class	Predicted class given the data
pred$posterior	Posterior probabilities

4.4 Training error and test error

```r
train_err <- mean(pred$class != train_data$y)
test_err <- mean(pred$class != test_data$y)
```

Or, create a confusion matrix:

```r
table(predicted = pred$class, actual = test_data$y)
```

4.5 Plot the Results

```r
ggplot(iris_test, aes(x=x1, y=x2, color=pred$class,
       shape=(pred$class == test_data$y))) +
    geom_point(size=1.5) +
    scale_shape_manual(values = c(4, 1),
                       breaks = c(TRUE, FALSE),
                       labels=c('Correctly classified',
```

```
                                  'Classification error'),
                        name='') +
    scale_color_discrete(name='Predicted class')
```

5 Logistic Regression

5.1 Model

glm: General Linear Model

```
mod1 <- glm(y ~ x1 + x2, family = binomial, data = D)
summary(mod1)
```

The output of the summary of the model 1 is:

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial, data = D)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6348  -0.8660  -0.4804   0.9793   2.4026
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -3.859709   0.489237  -7.889 3.04e-15 ***
## X1              0.060636   0.009827   6.170 6.81e-10 ***
## X2              1.010465   0.236172   4.279 1.88e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 506.95  on 392  degrees of freedom
## Residual deviance: 426.72  on 390  degrees of freedom
## AIC: 432.72
##
## Number of Fisher Scoring iterations: 4
```

5.2 Interaction

```
mod2 <- glm(y ~ x1 + x2 + x1:x2, family = binomial, data = D)
```

5.3 Variables and Functions

coef(mod1)[:]	Coefficients
pred <- predict(mod1, data)	Predict using the model (log-odds)
binomial()$linkinv(pred)	Function *expit*

6 Subset Selection

6.1 Model

```
library(leaps)
mod1 <- regsubsets(Y ~ ., data = D, nvmax = ncol(D) - 1)
mod2 <- regsubsets(Y ~ ., data = D, nvmax = ncol(D) - 1, method = "forward")
mod3 <- regsubsets(Y ~ ., data = D, nvmax = ncol(D) - 1, method = "backward")
```

6.2 Summary

```
summary(mod1)
```

The example output is:

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = htrain, nvmax = ncol(htrain) -
##     1)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: exhaustive
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns
## 1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "
## 2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 3  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 4  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "
## 6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "
## 7  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   "*"    "*"   "*"    " "
## 8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "
## 12  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    "*"
## 13  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    " "   " "    "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    " "   " "    "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    " "   " "    "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    " "   " "    "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"
##           CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
```

```
## 1  ( 1 )  "*"  " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 )  "*"  " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 )  "*"  " "    " "     " "       " "     " "     " "    " "
## 4  ( 1 )  "*"  " "    " "     " "       "*"     " "     " "    " "
## 5  ( 1 )  "*"  " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 )  "*"  " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 )  " "  " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 )  " "  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 )  " "  "*"    " "     "*"       "*"     "*"     " "    " "
## 10  ( 1 ) " "  "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) " "  "*"    " "     "*"       "*"     "*"     "*"    " "
## 12  ( 1 ) " "  "*"    " "     "*"       "*"     "*"     "*"    " "
## 13  ( 1 ) "*"  "*"    " "     "*"       "*"     "*"     "*"    " "
## 14  ( 1 ) "*"  "*"    " "     "*"       "*"     "*"     "*"    " "
## 15  ( 1 ) "*"  "*"    " "     "*"       "*"     "*"     "*"    " "
## 16  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 17  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    " "
## 19  ( 1 ) "*"  "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

If assigned to a variable, more information is contained.

which	Indicator matrix of variables included in the best model of a given size
rsq	R^2 statistics
rss	RSS statistics
adjr2	Adjusted R^2 statistics
cp	Mallow's C_p (equivalent to AIC)
bic	BIC statistics

6.3 Criteria

Summarize the statistics with respect to the dimension.

```
library(tidyverse)
bsub_data <-
  data.frame(bsub_summary[c('rss', 'adjr2', 'cp', 'bic')])
bsub_data$d <- rowSums(bsub_summary$which) - 1 # don't count the intercept
```

```
##         rss     adjr2        cp        bic  d
## 1  27205218 0.3458150 84.174500  -79.42964  1
## 2  23095257 0.4419615 42.337103 -108.47665  2
## 3  21666155 0.4739508 29.094110 -116.54348  3
## 4  20767579 0.4933084 21.509778 -120.09162  4
## 5  19955915 0.5107248 14.852457 -123.11669  5
## 6  19456611 0.5206167 11.526820 -123.09069  6
## 7  19060501 0.5280514  9.301854 -122.06302  7
## 8  18632969 0.5363420  6.741743 -121.47990  8
## 9  18359262 0.5408687  5.822342 -119.24045  9
## 10 18167489 0.5433814  5.776870 -116.09845 10
## 11 18058933 0.5438175  6.618999 -112.00991 11
## 12 18002356 0.5429383  8.015542 -107.32175 12
## 13 17943302 0.5421133  9.385659 -102.66465 13
## 14 17898201 0.5409220 10.904602  -97.84605 14
```

```
## 15 17857837 0.5395963 12.474077  -92.97307 15
## 16 17845935 0.5375192 14.347128  -87.76597 16
## 17 17835332 0.5353866 16.234038  -82.54366 17
## 18 17825474 0.5332122 18.128892  -77.31266 18
## 19 17813390 0.5310736 20.000000  -72.10796 19
```

What is the best model for given criterion?

```
best_d <-
data.frame(criterion = c('rss','cp','bic','adjr2'),
            best_d = bsub_data$d[c(apply(bsub_data[,c('rss','cp','bic')], 2,
    which.min),
                                   which.max(bsub_data$adjr2))]
)
```

```
##   criterion best_d
## 1       rss     19
## 2        cp     10
## 3       bic      5
## 4     adjr2     11
```

Visualization.

```
ggplot(bsub_data_long, aes(x=d, y=val)) +
geom_line() +
geom_vline(aes(xintercept = best_d), color='red',
            data=best_d) +
facet_wrap(~criterion, scales='free')
```

6.4 Fit the Best Model

Which candidate model has the lowest C_p?

```
which.min(bsub_summary$cp)
```

What predictors does the best model contain?

```
select_aic <- bsub_summary$which[which.min(bsub_summary$cp), ]
```

Fit the best model.

```
X_bsub_aic <- model.matrix(fullmod)[, select_aic]
bestfit_aic <- lm(htrain$Salary ~ 0 + X_bsub_aic)
```

Note that:

- The name of the predictors in *select_ aic* does not correspond exactly to the original dataset. It might include a *(Intercept)* term, as well as many dummy variables for categorical predictors.
- The second line fits the $Y = X\beta$ model without the intercept (indicated by "0 +").

6.5 Evaluate the Best Model

predict() function cannot be used in this case, since the variable names have changed. We need to calculate the predicted values by hand.

To get the coefficients:

```
bhat_best <- coef(bestfit_aic)
# bhat_best <- coef(bsubsets_full, id=which.min(bsub_summary$cp))
```

The full code snippet for retrieving training and test error.

```
X_bsub_aic <- model.matrix(fullmod)[, select_aic]
bhat_best <- coef(bestfit_aic)
yhat_train <- X_bsub_aic %*% bhat_best
mse_train <- mean((htrain$Salary - yhat_train)^2)

Xtest_bsub_aic <- model.matrix(Salary ~ ., data=htest)[, select_aic]
yhat_test <- Xtest_bsub_aic %*% bhat_best
mse_test <- mean((htest$Salary - yhat_test)^2)
```

As is known from the theory, if $k = 1$, it is called validation set approach. If $k = n$, it is called LOOCV.

6.6 Forward / Backward Selection based on Significance Test

```
library(SignifReg)
Select.p.fwd = SignifReg(lm(Salary ~., data = Hitters), alpha = 0.05,
                         direction = "forward", correction = "None", trace = F)
Select.p.bwd = SignifReg(lm(Salary ~., data = Hitters), alpha = 0.05,
                         direction = "backward", correction = "None", trace = F
    )
Select.p.fwd
Select.p.bwd
```

7 Resampling Methods

7.1 k-fold CV

```
n <- nrow(saheart)
nfolds <- 5
permute_n <- sample(n)
fsize <- floor(n / nfolds)
fsizes_exact <- c(rep(fsize, nfolds-1), fsize+n-fsize*(n%/%fsize))
fold_labels <- rep(1:nfolds, fsizes_exact)
fold_indices <- split(permute_n, fold_labels)
str(fold_indices)
```

```
## List of 5
##  $ 1: int [1:92] 462 212 175 281 187 69 163 72 282 305 ...
##  $ 2: int [1:92] 236 95 134 146 201 76 18 70 269 52 ...
##  $ 3: int [1:92] 208 338 200 329 30 273 209 16 184 128 ...
##  $ 4: int [1:92] 350 371 411 119 403 234 366 21 85 196 ...
##  $ 5: int [1:94] 344 378 107 154 341 214 356 109 110 150 ...
```

Create the CV train and CV validation set.

```
cv_train <- saheart[-fold_indices[[i]], ]
cv_valid <- saheart[fold_indices[[i]], ]
```

Fit the model and estimate the error.

7.2 Plot CV Estimates

```
ggplot(data.frame(cv=cv_byk, kn=kneighbors_seq),
       aes(x=log(1/kn), y=cv)) +
    geom_line()+
    scale_x_continuous(breaks = log(1/seq(min(kneighbors_seq), max(kneighbors_
    seq), 5)),
       labels = seq(min(kneighbors_seq), max(kneighbors_seq), 5)) +
    xlab("K (# neighbors)") + ylab("CV error") +
    ggtitle(paste(nfolds, "-fold CV", sep=''))
```

7.3 Bootstrap

```
library(boot)
set.seed(108)
tau <- 0.2 # maximum probability of failure
logit_tau <- log(tau / (1-tau))

# Write a function with two arguments:
#   first argument: full data set
#   second argument: vector of indices defining each bootstrap sample
#  return value: the hat(theta)_b statistic for the bootstrap sample
theta_hat_fn <- function(data, ix) {
  model_b <- glm(fail ~ temp, family=binomial, data = data[ix, ])
  b0 <- coef(model_b)[1]
  b1 <- coef(model_b)[2]
  return((logit_tau - b0) / b1)
}

fit_boot <- boot(data= o2, statistic = theta_hat_fn, R=500)

str(fit_boot)

ggplot(data.frame(t=fit_boot$t),aes(x=t))+
  geom_histogram(bins=30, fill='white', color='black') +
  ggtitle("Bootstrap distribution of hat(theta) using boot package")
```

8 Shrinkage Methods

8.1 Preparation

```
X <- model.matrix(Y ~ ., D)[, -1]
y <- D$Y

set.seed(1029)
trainFrac <- 0.7
train_ix <- sample(nrow(X), floor(trainFrac * nrow(X)))
Xtrain <-  X[train_ix,]
ytrain <- y[train_ix]
```

8.2 Model

```
library(glmnet)
grid <- 10 ^ seq(10, -2, length=100)
ridge.mod <- glmnet(x=Xtrain, y=ytrain, alpha=0, lambda=grid, standardize=FALSE
    )
lasso.mod <- glmnet(x=Xtrain, y=ytrain, alpha=1, lambda=grid, standardize=FALSE
    )
```

8.3 Prediction

```
# Coefficients
coef(ridge.mod)

# The predicted y values for the test data when lambda=0.1
predict(ridge.mod, newx=X[-train_ix,], s=0.1, type="response")

# The predicted y values for the test data at every grid value of lambda
predict(ridge.mod, newx = X[-train_ix,], type='response')
```

8.4 Choosing λ by Cross-Validation

```
set.seed(1)
cv.ridge <- cv.glmnet(x=Xtrain, y=ytrain, alpha=0, lambda=grid)
```

```
bestlam_ridge <- cv.ridge$lambda.min

# training MSE
ridge.pred_train <- predict(ridge.mod, s=bestlam_ridge, newx=Xtrain)
train_MSE <- mean((ridge.pred_train - ytrain)^2)

# test MSE
ridge.pred_test <- predict(ridge.mod, s=bestlam_ridge, newx=X[-train_ix,])
test_MSE <- mean((ridge.pred_test-y[-train_ix])^2)

# CV MSE
which(cv.ridge$lambda == bestlam_ridge)
cv.ridge$cvm[which(cv.ridge$lambda == bestlam_ridge)]
```

8.5 Visualization

Coefficients-Lambda Relationship

```
plot(ridge.mod, xvar = "lambda", label = TRUE)
```

CV-MSE versus Lambda

```
plot(cv.ridge)
```

9 Dimension Reduction

9.1 Principal Component Analysis

```
# Data matrix without intercept
X <- model.matrix(Y ~ ., data = D)[, -1]

# Run PCA
Xpca <- prcomp(x = X, center = TRUE, scale = TRUE)

# Variance Contribution
summary(Xpca)
pc_stddev = Xpca$sdev

# Loadings
pc_loadings = Xpca$rotation

# Scores
pc_scores = predict(Xpca)

# Scree plot
plot(Xpca)
```

```
## Importance of components:
##                           PC1    PC2     PC3    PC4     PC5
## Standard deviation     2.3243 1.9063 1.11090 0.9608 0.90235
## Proportion of Variance 0.3377 0.2271 0.07713 0.0577 0.05089
## Cumulative Proportion  0.3377 0.5648 0.64190 0.6996 0.75048
##                           PC6     PC7     PC8     PC9    PC10
## Standard deviation     0.88719 0.80575 0.75502 0.70909 0.62890
## Proportion of Variance 0.04919 0.04058 0.03563 0.03143 0.02472
## Cumulative Proportion  0.79968 0.84025 0.87588 0.90731 0.93203
##                           PC11    PC12    PC13    PC14   PC15
## Standard deviation     0.58175 0.55558 0.42608 0.37996 0.3021
## Proportion of Variance 0.02115 0.01929 0.01135 0.00902 0.0057
```

```
## Cumulative Proportion  0.95318 0.97247 0.98382 0.99284 0.9986
##                            PC16
## Standard deviation     0.15255
## Proportion of Variance 0.00145
## Cumulative Proportion  1.00000
```

9.2 Principal Component Regression

```r
# Library and Model
library(pls)
set.seed(1)
mod_pcr = pcr(Y ~ ., data = D, subset = train_id,
              scale = TRUE, validation = "CV", segments = 10)
summary(mod_pcr)

# CV RMSE plot versus k
validationplot(mod_pcr, val.type = "MSEP", legendpos = "topright")

# Choose the best k
MSEP(mod_pcr)$val["CV", "Y", ]
best_CV_error <- min(MSEP(mod_pcr)$val["CV","Y",])
best_m <- (0:ncol(train_set))[which.min(MSEP(mod_pcr)$val["CV","Y",])]

# Training error
pcr_pred_train <- predict(mod_pcr, train_set, ncomp = best_m)
train_MSE_pcr <- mean((pcr_pred_train - train_set$Y)^2)

# Test error
pcr_pred_test <- predict(mod_pcr, test_set, ncomp = best_m)
test_MSE_pcr <- mean((pcr_pred_test - test_set$Y)^2)

# Original predictor coefficients
mod_pcr$coefficients[, , 1]
```

9.3 Partial Least Squares

```r
# Library and model
library(pls)
set.seed(1)
mod_pls <- plsr(Y ~ ., data = D, subset = train_id,
                scale = TRUE, validation = "CV", segments = 10)
summary(mod_pls)

# Choose the best k
MSEP(mod_pls)$val["CV", "Y", ]
best_CV_error <- min(MSEP(mod_pls)$val["CV","Y",])
best_m <- (0:ncol(train_set))[which.min(MSEP(mod_pls)$val["CV","Y",])]

# Training error
pls_pred_train <- predict(mod_pls, train_set, ncomp = best_m)
train_MSE_pls <- mean((pls_pred_train - train_set$Y)^2)

# Test error
pls_pred_test <- predict(mod_pls, test_set, ncomp = best_m)
test_MSE_pls <- mean((pls_pred_test - test_set$Y)^2)
```

9.4 Comparison between Models

When running multiple CV models, the random seed should be RESET if necessary.

```r
d <- data.frame("TestMSE" = c(PCRTestMSE, PLSTestMSE, ridgeTestMSE,
    lassoTestMSE))
```

```r
rownames(d) <- c("PCR", "PLS", "Ridge", "Lasso")
knitr::kable(d)
```

10 Decision Trees

10.1 Classification Trees

Basics

```r
# Library and model
library(tree)
mod_tree = tree(Y ~ ., Data)

# Text summary of the tree
summary(mod_tree)

# Plot the classification tree
plot(mod_tree)
text(mod_tree, pretty = 0, cex = 0.5)

# Evaluation
train.pred = predict(mod_tree, train_set, type = "class")
train.err = mean(train.pred != train_set$Y)
table(train.pred, train_set$Y)
test.pred = predict(mod_tree, test_set, type = "class")
test.err = mean(test.pred != test_set$Y)
table(test.pred, test_set$Y)
```

Pruning

```r
set.seed(1)
cv_mod_tree = cv.tree(mod_tree, FUN = prune.misclass)
cv_mod_tree
```

```
## $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"
```

- `dev` is the number of misclassification for the K-fold CV procedure.
- `k` is the tuning parameter α
- `size` is the tree size (leaves)

```r
# CV error
cv_err <- cv_mod_tree$dev / nrow(train_set)

# Pruning the tree
prune.tree <- prune.misclass(mod_tree, best = cv_mod_tree$size[which.min(cv_mod
    _tree$dev)])
plot(prune.tree)
text(prune.tree, pretty = 0)

# Test error
test.pred = predict(prune.tree, test_set, type = "class")
test.err = mean(test.pred != test_set$Y)
table(test.pred, test_set$Y)
```

10.2 Regression Trees

```r
# Library and model
library(tree)
mod_tree = tree(Y ~ ., Data, subset = train_id)
summary(mod_tree)

# Plot the regression tree
plot(mod_tree)
text(mod_tree, pretty = 0)

# Cross-validation
cv_mod_tree = cv.tree(mod_tree)
cv_mod_tree

# Plot deviance against size
plot(cv_mod_tree$size, cv_mod_tree$dev, type = 'b')

# Pruning
prune_mod = prune.tree(mod_tree, best = cv_mod_tree$size[which.min(cv_mod_tree$
    dev)])
plot(prune_mod)
text(prune_mod, pretty = 0)

# Evaluation
test.pred = predict(prune_mod, newdata = test_set)
test.mse = mean((test.pred - test_set$Y)^2)
```

11 Ensemble Methods

11.1 Bagging

```r
library(randomForest)
mod.bagging <- randomForest(Y ~ ., data = D, subset = train,
                            ntree = 500, mtry = ncol(D) - 1, importance = TRUE)
mod.bagging
```

The argument `mtry = ncol(Boston) - 1` indicates that all predictors should be considered for each split of the tree. `mtry` can be ignored.

```
##
## Call:
##  randomForest(formula = Y ~ ., data = D, mtry = ncol(Boston) - 1,
##               ntree = 100, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
```

```
##
##           Mean of squared residuals: 11.15723
##                     % Var explained: 86.49
```

```r
# Evaluation
train.pred.bag = predict(mod.bagging, newdata = train_set)
train.mse.bag = mean((train.pred.bag - train_set$Y)^2)
test.pred.bag = predict(mod.bagging, newdata = test_set)
test.mse.bag = mean((test.pred.bag - test_set$Y)^2)

# Visualization
plot(test.pred.bag, test_set$Y)
abline(0, 1, col = "red")
```

11.2 Random Forests

```r
library(randomForest)
mod.forest <- randomForest(Y ~ ., data = D, subset = train,
                           ntree = 500, mtry = 6, importance = TRUE)
mod.forest
```

Default value for `mtry`: $p/3$ for regression trees and $\sqrt{p}$ for classification trees

```r
# Relative Variable Importance
importance(mod.forest)
varImpPlot(mod.forest)
```

11.3 Gradient Boosting

```r
library(gbm)
mod.boost <- gbm(Y ~ ., data = train_set, distribution = "gaussian",
                 n.trees = 5000, interaction.depth = 4, shrinkage = 0.2,
                 cv.folds = 5)
summary(mod.boost)    # variable importance
```

- `n.trees` indicates the number of trees
- `interaction.depth` limits the depth of each tree

```r
# Partial dependence plots for certain variable
plot(mod.boost, i = "var")
```

Prediction and evaluation:

```r
best_m <- which.min(boost.boston$cv.error)

train.pred.boost = predict(mod.boost, newdata = train_set, n.trees = best_m)
train.mse.boost = mean((train.pred.boost - train_set$Y)^2)
test.pred.boost = predict(mod.boost, newdata = test_set, n.trees = best_m)
test.mse.boost = mean((test.pred.boost - test_set$Y)^2)
```

11.4 AdaBoost

```r
# Code the variable in binary form.
D$Y2 = ifelse(D$Y == "A", 1, 0)

# Library and Model
library(gbm)
mod.ada <- gbm(Y ~ ., data = train_set, distribution = "adaboost",
               n.trees = 1000)
summary(mod.ada)    # variable importance
```

```
9
10 # Evaluation
11 train.pred.ada = predict(mod.ada, newdata = train_set, n.trees = 1000, type = "
       response")
12 train.mse.ada = mean(train.pred.ada != train_set$Y)
13 test.pred.ada = predict(mod.ada, newdata = test_set, n.trees = 1000, type = "
       response")
14 test.mse.ada = mean(test.pred.ada != test_set$Y)
```

12 Support Vector Machines (SVM)

12.1 Support Vector Classifier

Model

```
1 library(e1071)
2 mod.svm <- svm(Y ~ ., data = train_set, kernel = "linear",
3                cost = 10, scale = FALSE)
4 # The response must be a factor to do classification
5 # cost controls penalization
```

Visualization

```
1 plot(mod.svm, data = train_set, formula = X2 ~ X1)
```

- `data` gives the data we want to plot
- `formula` specifies the X and Y axis

Summary Information

```
1  summary(mod.svm)
2  mod.svm$index      # which observations are the support vectors
3  mod.svm$coefs      # hat(alpha)_i * y_i for the support vectors
4  mod.svm$rho        # -beta0
5
6  alpha_times_y <- mod.svm$coefs          # hat(beta) = sum_i hat(alpha)_i y_i x_
       i
7  betahat <- as.numeric(t(alpha_times_y) %*% as.matrix(dat[mod.svm$index,c(1,2)])
       )
8  beta0 <- -mod.svm$rho
9
10 ggplot(cbind(dat,
11              suppvec = 1:nrow(dat) %in% mod.svm$index # label the support
       vectors
12              ),
13        aes(x.1, x.2, color=y, shape=suppvec)) +
14   geom_point(size=2) +
15   geom_abline(aes(slope=-betahat[1] / betahat[2], # decision boundary
16                   intercept=-beta0 / betahat[2] )) +
17   geom_abline(aes(slope=-betahat[1] / betahat[2], # margin
18                   intercept=1/betahat[2] - beta0 / betahat[2] ),
19               linetype='dashed') +
20   geom_abline(aes(slope= -betahat[1] / betahat[2], # margin
21                   intercept= -1/betahat[2] - beta0 / betahat[2] ),
22               linetype='dashed') +
23   scale_shape_manual(values=c('FALSE'=1,'TRUE'=4)) +
24   theme(panel.background = element_rect(fill=NA,color='lightgrey'))+
25   scale_color_brewer(palette='Set1') +
26   coord_fixed()
```

Cross Validation

```r
mod.tune.svm <- tune(svm, Y ~ ., data = train_set, kernel = "linear",
                     ranges = list(cost = c(0.001,0.01,0.1,1,5,10,100)))
summary(mod.tune.svm)

# Access the best model
best.mod <- mod.tune.svm$best.model
```

```
##
## 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
```

Evaluation

```r
train.pred.svm = predict(mod.svm, train_set)
train.mse.svm = mean(train.pred.svm != train_set$Y)
test.pred.svm = predict(mod.svm, test_set)
test.mse.svm = mean(test.pred.svm != test_set$Y)

table(predict = test.pred.svm, truth = test_set$Y)
```

12.2 Support Vector Machines

```r
mod.svm2 <- svm(Y ~ ., data = train_set, kernel = "radial", gamma = 1, cost =
    1)
# kernel = polynomial with a degree argument
# kernel = radial with a gamma argument
```

Cross Validation

```r
# Tuning
tune.out <- tune(svm, Y ~ ., data = train_set, kernel = "radial",
                 ranges = list(cost = c(0.1, 1, 10, 100, 1000),
                 gamma = c(0.5, 1, 2, 3, 4)))
tune.out$best.parameters
bestmod <- tune.out$best.model
summary(bestmod)

# Visualization (base-R)
with(tune.out$performances, {
  plot(error[gamma == 0.5] ~ cost[gamma == 0.5], ylim = c(.1, .35),
       type = "o", col = rainbow(5)[1], ylab = "CV error", xlab = "cost")
```

```
  lines(error[gamma == 1] ~ cost[gamma == 1],
        type = "o", col = rainbow(5)[2])
  lines(error[gamma == 2] ~ cost[gamma == 2],
        type = "o", col = rainbow(5)[3])
  lines(error[gamma == 3] ~ cost[gamma == 3],
        type = "o", col = rainbow(5)[4])
  lines(error[gamma == 4] ~ cost[gamma == 4],
        type = "o", col = rainbow(5)[5])
})
legend("bottom", horiz = T, legend = c(0.5, 1:4), col = rainbow(5),
       lty = 1, cex = .75, title = "gamma")

# Visualization (ggplot)
tuneSummary <- summary(tune.out)
ggplot(tuneSummary$performances,
       aes(x=log(cost), y=error,color=as.factor(gamma)))+
  geom_point() + geom_line()+xlab("cost")+ylab("CV error")+
  scale_color_brewer(palette='Set1')+
  scale_x_continuous(breaks=log(10^seq(-1,2,length.out=7)),
                     labels=round(10^seq(-1,2,length.out=7),3))
```

12.3 Multiple Classes - One-Versus-One Classification

The same argument as in previous section, except that the factor variable y consists of more than 2 levels.

13 Clustering

Some assistant functions to plot the clusters:

```
library(tidyverse)
# Function to plot all
# pairs of variables in the iris data set,
# coloring the points according to a provided vector
plot_iris_pairs <- function(color_vec = NULL){
    mytheme <- theme(panel.background = element_rect(fill=NA,color='grey'),
                     strip.background = element_rect(fill=NA,color='grey'),
                     axis.text=element_blank())

    i2 <- iris %>% mutate(ix=row_number())
    if (!is.null(color_vec)) {
    i2 <- i2 %>% mutate(clab=color_vec)
    iris_long <-
        i2 %>%
        gather(-ix, -clab, -Species, key='var',value='value')
    } else{
    iris_long <-
        i2 %>%
        gather(-ix, -Species, key='var', value='value')
    }

    dpairs <- lapply(combn(unique(iris_long$var), 2, simplify=FALSE),
        function(vc) return(filter(iris_long, var %in% vc) %>%
                                mutate(var = factor(var, levels=unique(var),
                                                    labels=c('v1','v2')))%>%
                                spread(var, value)))
    names(dpairs) <- do.call('c',
                             lapply(
                                 combn(unique(iris_long$var), 2, simplify=FALSE)
    ,
                                 function(x) return(paste(x,collapse=', '))))
    ret <-
    bind_rows(dpairs, .id='pair')%>%
```

```
    ggplot(aes(x=v1,y=v2, shape=Species))
    if (!is.null(color_vec)){
    ret <- ret + geom_point(aes(color=as.factor(clab)))+
        scale_color_brewer(palette='Set1',name='Cluster')
    } else{
    ret <-
    ret + geom_point(size=1.2)
    }
    ret <- ret + mytheme +
    xlab("")+ylab("")+
    scale_shape_manual(values=c(0,1,3))+
    facet_wrap(~pair, scales='free')
    return(ret)
}

plot_iris_pairs(iris$Species) +
    ggtitle("True clusters: species")
```

13.1 Hierarchical Clustering

```
# Prepare the matrix
X <- as.matrix(D[,c('X1','X2','X3')])
set.seed(416)

# Euclidean distances between each pair of points
(Xdist <- dist(X))

# Use different dissimilarity measures b/t groups
hc.average <- hclust(Xdist, method='average')
hc.single <- hclust(Xdist, method='single')
hc.complete <- hclust(Xdist, method='complete')

# Dendrogram
par(mfrow=c(1,3))
plot(hc.complete,main="Complete Linkage", xlab="", sub="", cex=.9)
plot(hc.average, main="Average Linkage", xlab="", sub="", cex=.9)
plot(hc.single, main="Single Linkage", xlab="", sub="", cex=.9)

# Heights
hc.complete$height

# Cut the tree
cutree(hc.complete, h = 0)      # cut at height = 0
cutree(hc.complete, k = 3)      # cut to obtain 3 clusters

# Visualization
grid.arrange(
  plot_iris_pairs(cutree(hc.complete, 3)) +
  ggtitle("3 clusters, complete linkage"),
  plot_iris_pairs(cutree(hc.average, 3))+
  ggtitle("3 clusters, average linkage"),
  plot_iris_pairs(cutree(hc.single, 3))+
  ggtitle("3 clusters, single linkage"),
  plot_iris_pairs(iris$Species) +
    ggtitle("True clusters"),
  nrow=2)
```

13.2 K-Means Clustering

```
# Model
set.seed(1)
km.out <- kmeans(X, centers = 2)    # centers argument specifies K
```

```r


# Quantities
km.out$centers           # Centroids
km.out$cluster           # Clusters ID of each point
km.out$total.withinss    # Within-cluster sum of square


# Visualization: Plot the iterations
# Function to plot cluster means
pairs_iris_cmeans <- function(cmeans){
  d <- as_tibble(cmeans) %>%
    mutate(clab=row_number())
  dlong <- d %>% gather(-clab, key='var',value='value')
  dpairs <- lapply(combn(unique(dlong$var), 2, simplify=FALSE),
       function(vc) return(filter(dlong, var %in% vc) %>%
                             mutate(var = factor(var, levels=unique(var),
                                                 labels=c('v1','v2')))%>%
                             spread(var, value)))
  names(dpairs) <- do.call('c',
                           lapply(
                             combn(unique(dlong$var), 2, simplify=FALSE),
                             function(x) return(paste(x,collapse=', '))))
  return(
    bind_rows(dpairs, .id='pair')
  )
}

library(MASS)
set.seed(120)
# Random starting value near the point cloud
startHere <- mvrnorm(n=3, mu=c(4.5, 3.1, 3, 0.75), Sigma=0.08*diag(4))
colnames(startHere) <- colnames(X)
# Fit each iteration
kf_i1 <- kmeans(X, startHere, iter.max=1)
kf_i2 <- kmeans(X, kf_i1$centers, iter.max=1)
kf_i3 <- kmeans(X, kf_i2$centers, iter.max=1)

grid.arrange(
  plot_iris_pairs() +
    geom_point(size=4, stroke=2,
               shape=1,color='black',
               data=pairs_iris_cmeans(startHere)) +
    geom_point(aes(color=as.factor(clab)),
               size=3,
               shape=19,
               data=pairs_iris_cmeans(startHere)) +
  scale_color_brewer(palette='Set1')+ggtitle("Starting point"),
  plot_iris_pairs(kf_i1$cluster) +
    geom_point(size=4, stroke=2,
               shape=1,color='black',
               data=pairs_iris_cmeans(kf_i1$centers)) +
    geom_point(aes(color=as.factor(clab)),
               size=3, shape=19,
               data=pairs_iris_cmeans(kf_i1$centers)) +ggtitle("First iteration")
    ,
  plot_iris_pairs(kf_i2$cluster) +
    geom_point(size=4, stroke=2,
               shape=1,color='black',
               data=pairs_iris_cmeans(kf_i2$centers)) +
    geom_point(aes(color=as.factor(clab)),
               size=3, shape=19,
               data=pairs_iris_cmeans(kf_i2$centers)) +ggtitle("Second iteration"
```

```
  ),
plot_iris_pairs(kf_i3$cluster) +
  geom_point(size=4, stroke=2,
             shape=1,color='black',
             data=pairs_iris_cmeans(kf_i3$centers)) +
  geom_point(aes(color=as.factor(clab)),
             size=3,shape=19,
             data=pairs_iris_cmeans(kf_i3$centers)) + ggtitle("Third iteration"
  ),
nrow=2)
```