```
###Linear Regression:
#fitting the model
lm.fit <- lm(medv ~lstat, data = Boston)
# finding the coefficients of the model
coef(lm.fit)
#finding the confidence interval of the model
confint(lm.fit)
# finding the confidence and prediction
intervals of predicted points
predict(lm.fit, data.frame(lstat = c(5,10,15)),
interval = "confidence")
predict(lm.fit, data.frame(lstat = c(5,10,15)),
interval = "prediction")
#plotting points against regression line
plot(lstat, medv)
abline(lm.fit)
#plotting the residuals
par(mfrow = c(2,2))
plot(lm.fit)
plot(predict(lm.fit), residuals(lm.fit))
plot(predict(lm.fit), rstudent(lm.fit))
plot(hatvalues(lm.fit)) # leverage values
which.max(hatvalues(lm.fit)) # highest leverage
#fit using all the values
lm.fit = lm(medv~ . , data = Boston)
#printing values from summary
?summary.lm
summary(lm.fit)$r.sq
# looking at VIF of model
library(car)
vif(lm.fit)
#adding in interactionterm(auto adds in
original terms)
summary(lm(medv~lstat*age, data = Boston))
#adding in non linearity
lm.fit2 = lm(medv~lstat+I(lstat^2))
lm.fit5 = lm(medv~poly(lstat,5))
#comparing models – p value for if 2nd model is
better
anova(lm.fit, lm.fit2)
# shows how the model will look at dummy vars
contrasts(ShelveLoc)
#fitting linear model with no intercept
lm(y~x+0)

###Logistic Regression:
#fitting the model
glm.fits = glm(Direction ~
Lag1+Lag2+Lag3+Lag4+Lag5+Volume, family =
"binomial", data= Smarket)
#summary of the model
summary(glm.fits)
summary(glm.fits)$coef[,4]
#checking what is set to 0 and 1
contrasts(Direction)
#predicting the probabilities
glm.probs = predict(glm.fits, type =
"response")
#predicting classes
glm.pred = rep("Down", nrow(Smarket))
glm.pred[glm.probs >0.5] = "Up"
#confusion matrix
table(glm.pred,Direction)
# classification percentage
mean(glm.pred == Direction)
# train and test
train =(Year < 2005)
Smarket.2005 = Smarket[!train,]
Direction.2005= Smarket$Direction[!train]
#fitting on training set
glm.fits = glm(Direction ~
Lag1+Lag2+Lag3+Lag4+Lag5+Volume, family =
"binomial", data= Smarket, subset = train)
#predicting on the test set
glm.probs = predict(glm.fits, Smarket.2005,
type = "response")

###Linear Discriminant Analysis:
library(MASS)
#fitting the model
lda.fit = lda(Direction ~ Lag1+Lag2, data =
Smarket, subset = train)
#plotting the linear discriminants for each
index
plot(lda.fit)
#predicting
lda.pred = predict(lda.fit, Smarket.2005)
#class, posterior,x
names(lda.pred)
lda.class = lda.pred$class

###Variable Selection:
library(leaps)
#best subset
regfit.full = regsubsets(Salary~ ., data =
Hitters)
#can set number of variables
regfit.full = regsubsets(Salary~ ., data =
Hitters, nvmax=19)
reg.summary = (summary(regfit.full))
#summary of stats over all models
names(reg.summary)
reg.summary$rsq
#plotting stats over number of variables
par(mfrow = c(2,2))
plot(reg.summary$rss, xlab = "Number of
Variables", ylab = "RSS", type = 'l')
plot(reg.summary$adjr2, xlab = "Number of
Variables", ylab = "Adjusted R2", type = 'l')
#plotting max point
which.max(reg.summary$adjr2)
points(11,reg.summary$adjr2[11], col="red",
cex=2, pch = 20)
plot(reg.summary$cp, xlab = "Number of
Variables", ylab = "Cp", type = 'l')
which.min(reg.summary$cp)
points(10,reg.summary$cp[10], col="red", cex=2,
pch = 20)
plot(reg.summary$bic, xlab = "Number of
Variables", ylab = "BIC", type = 'l')
which.min(reg.summary$bic)
points(6,reg.summary$bic[6], col="red", cex=2,
pch = 20)
#coeff of model with x no variables
coef(regfit.full, 6)
#forward and backward
regfit.fwd = regsubsets(Salary ~ ., data =
Hitters, nvmax = 19, method = "forward" )
regfit.bwd = regsubsets(Salary ~ ., data =
Hitters, nvmax = 19, method = "backward" )
#coeff of diff models with same no of variables
coef(regfit.full,7)
coef(regfit.fwd,7)
coef(regfit.bwd,7)
set.seed(1)
# Another good example of regsubset
x <- rnorm(100)
eps <- rnorm(100)
b0 <- 2
b1 <- 3
b2 <- -1
b3 <- 0.5
y <- b0 + b1 * x + b2 * x^2 + b3 * x^3 + eps
library(leaps)
data.full <- data.frame(y = y, x = x)
regfit.full <- regsubsets(y ~ x + I(x^2) +
I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) +
I(x^8) + I(x^9) + I(x^10), data = data.full,
nvmax = 10, method = "exhaustive")
reg.summary <- summary(regfit.full)
par(mfrow = c(2, 2))
plot(reg.summary$cp, xlab = "Number of
variables", ylab = "C_p", pch = 19, type = "b")
points(which.min(reg.summary$cp),
reg.summary$cp[which.min(reg.summary$cp)], col
= "red", cex = 2, pch = 20)
plot(reg.summary$bic, xlab = "Number of
variables", ylab = "BIC", type = "l")
points(which.min(reg.summary$bic),
reg.summary$bic[which.min(reg.summary$bic)],
col = "red", cex = 2, pch = 20)
plot(reg.summary$adjr2, xlab = "Number of
variables", ylab = "Adjusted R^2", type = "l")
points(which.max(reg.summary$adjr2),
reg.summary$adjr2[which.max(reg.summary$adjr2)]
, col = "re d", cex = 2, pch = 20)
coef(regfit.full, which.max(reg.summary$adjr2))

###Validation and Cross Validation
  #Validation
  train = sample(c(TRUE,FALSE), nrow(Hitters),
rep = T)
test = (!train)
#fitting on train data
regfit.best = regsubsets(Salary~ . , data =
Hitters[train,], nvmax= 19)
# test set with dummy variables sorted
test.mat = model.matrix(Salary~., data =
Hitters[test,])
#validation errors for different sizes
val.errors = rep(NA,19)
for (i in 1:19){
  coefi = coef(regfit.best, id = i)
  pred = test.mat[,names(coefi)]%*%coefi
  val.errors[i] = mean((Hitters$Salary[test] -
pred)^2)}
#minimum validation error
which.min(val.errors)
#predict function
predict.regsubsets = function(object, newdata,
id, ...){
  form = as.formula(object$call[[2]])
  mat = model.matrix(form, newdata)
  coefi = coef(form, id = id)
  print(coefi)
  xvars = names(coefi)
  mat[,xvars]%*%coefi}
#cross validation
k = 10
folds = sample(1:k, nrow(data), replace = TRUE)
cv.errors = matrix(NA,k,19, dimnames =
list(NULL, paste(1:19)))
for (j in 1:k){
  best.fit = regsubsets(Salary~.,data =
data[folds != j,], nvmax=19)
  for (i in 1:19) {
    pred = predict.regsubsets(best.fit,
data[folds == j,], id = i)
    cv.errors[j,i] = mean((data$Salary[folds ==
j]- pred)^2)}}
#mean over the errors
mean.cv.errors = apply(cv.errors,2,mean)
#plotting errors vs no. of variables
plot(mean.cv.errors, type = 'b')
reg.best = regsubsets(Salary~., data = data,
nvmax = 19)
coef(reg.best,11)

###Regularization:
x <- model.matrix(lgRate ~ . , RoadData)[, -1]
x <- scale(x)
y <- RoadData$lgRate
library(glmnet)
#use family = "binomial for logistic regression
# lasso: alpha = 1, ridge: alpha = 0
lasso.mod <- glmnet(x,y,alpha = 1)
plot(lasso.mod)
# plot variable coefficients vs lambda
plot(lasso.mod, xvar = "lambda")
# can add argument type.measure
cv.out = cv.glmnet(x,y, alpha = 1)
plot(cv.out)
bestlam = cv.out$lambda.min
sprintf("MSE for Lasso model: %s", bestlam)
lasso.pred = predict(lasso.mod, s = bestlam,
newx = as.matrix(test))
lasso.coeff = predict(lasso.mod, type =
"coefficients", s = bestlam)
print(lasso.coef)
print(lasso.coef[lasso.coef != 0])
#KNN
train.X <- as.matrix(Insurance.train[,-1])
train.Y <- as.matrix(Insurance.train[,1])
test.X <- as.matrix(Insurance.test[,-1])
library(class)
pred.knn <- knn(train.X, test.X, train.Y, k =
1)

###Extras:
Hitters = na.omit(Hitters)
set.seed(1)
fulldata <- read.csv("my_boston.csv")
Stratified sampling
combined = cbind(data_unlab$ID,yhat_unlab)
colnames(combined) = c("ID", "Class")
write.csv(combined, file = "unlabeledpred.csv",
row.names = FALSE)
lsf.str("package:dplyr") # view functions in
package
## stepwise selection for logistic regression
Library(MASS)
null = glm(spam ~ 1, data = train, family =
"binomial")
mod = glm(spam ~ 1, data = train, family =
"binomial")
step = stepAIC(null, trace = F, direction =
"both", scope = formula(mod))
forward = stepAIC(null, direction = "forward",
scope = formula(mod), trace = F)
backward = stepAIC(mod, direction =
"backwards", trace = F)
##stepwise selection for DA
Library(klaR)
Both = stepclass(spam ~ ., data = train, method
= "lda", start.vars = "A.1", maxvar = 57) #
direction = "forward"
Backward = stepclass(spam ~ ., data = train,
method = "lda",  direction = "backward")
### with probability values
glm.fit <- glm(Class ~ ., data =
Insurance.train, family = binomial())

probs <- predict(glm.fit, newdata =
Insurance.test, type = "response")
glm.pred <- rep(1, length(probs))
glm.pred[probs > 0.5] <- 2
```

```
confusionMatrix(factor(glm.pred),
Insurance.test$Class, mode = "everything")
table(glm.pred, Insurance.test$Class)
plot(probs, as.factor(glm.pred), col =
(Insurance.test$Class != 1) +9, type="p",
xlab="probability", ylab="Responce", yaxt="n")
grid()
abline(v=c(0.5), lty=2, lwd=3)
axis(2, at=1:2, labels=c('Not
fraudulent','fraudulent'))

###Trees:
library(tree)
tree_model <- tree(medv ~ . , fulldata[,-1],
subset = split.data$train,
control=tree.control(length(split.data$train),m
insize = 20, mincut = 10, mindev = 0.0001))
yhat = predict(tree_model, newdata = fulldata[-
split.data$train,-1])
print(mean((yhat-boston.test)^2))
print(tree_model)
summary(tree_model)
plot(tree_model)
text(tree_model, pretty = 0)
##alphas and the deviance associated for the
subtrees
alphas = prune.tree(tree_model)$k
plot(prune.tree(tree_model))
cv.boston = cv.tree(tree_model, K =10)
print(cv.boston)
plot(cv.boston$size, cv.boston$dev, type = 'b',
col = 'blue', ylim = c(2000,17000))
lines(prune.tree(tree_model)$size,
prune.tree(tree_model)$dev, col = "red", type =
'b')
legend("topright", inset = .05, c("cv", "full
dataset"), fill = c("blue","red"))
## pruning the tree
final.boston = prune.tree(tree_model, best = 6)
plot(final.boston)
text(final.boston, pretty = 0)
##Selecting best tree size from fully grown
tree
stopcrit <-
tree.control(nobs=nrow(mushrooms.train),
minsize = 1, mindev = 0)
treemodel = tree(poison ~ ., data =
mushrooms.train, control = stopcrit,
split="deviance")
plot(treemodel)
text(treemodel, pretty = 0)
probs <- predict(treemodel, newdata =
mushrooms.test)
tree.pred <- rep('e', dim(probs)[1])
tree.pred[probs[,2]> 0.5] <- 'p'
tree.pred <- as.factor(tree.pred)
cv.tree = cv.tree(treemodel,FUN=prune.misclass,
K = 10)
plot(cv.tree$size ,cv.tree$dev ,type="b")
best <- which.min(rev(cv.tree$dev))
prunedtree = prune.tree(treemodel, best)

### RF without H2o
#if mtry = number of predictors then we have
bagging.
library(randomForest)
randomForest = randomForest(poison ~ ., data =
mushrooms.train, mtry = 3, ntree = 100,
importance = TRUE)
randomForest = randomForest(poison ~ ., data =
mushrooms.train, mtry =
sqrt(ncol(mushrooms.train)), ntree = 100,
importance = TRUE)
varImpPlot(randomForest, type=2)
# Boosed trees without h2o
library(gbm)
boosting <- gbm(poison ~ ., data =
mushrooms.train, cv.folds = 10,
distribution="gaussian")
summary(boosting)
plot(boosting$cv.error)

###RF without grid h2o
library(h2o)
rf.spam <- h2o.randomForest(
  training_frame = as.h2o(mushrooms.train), y =
1, seed = 1)
rf.probs <- h2o.predict(
  object = rf.spam,
  newdata = as.h2o(mushrooms.test)
)
rf.probs <- as.data.frame(rf.probs)



##RF with grid h2o
## initializing variables for grid search
mtries <- c(2,3,4)
ntrees = c(25,50,75,100,125)
max_depth = c(4,5,6,7,8,9)
hyper_params <- list(mtries = mtries, ntrees =
ntrees, max_depth = max_depth)
## running grid search
rf1_grid <- h2o.grid(algorithm =
"randomForest",
                     hyper_params =
hyper_params,
                     x = 2:58, y = 59,
                     training_frame = spam_h2o,
                     #validation_frame =
datTest_h2o,
                     nfolds = 0,
                     seed = 1)
summary(rf1_grid)
## choosing best model
best_rf <-
h2o.getModel(rf1_grid@model_ids[[1]])
perf_rf <- h2o.performance(best_rf, newdata =
spam_h2o)
auc.val <- h2o.auc(best_rf)
## variable importance plot
h2o.varimp_plot(best_rf)
#prediction
predictionsTest <- h2o.predict(best_rf,
test_h2o)
yhatTest = (as.matrix(predictionsTest$spam))
## partial dependence plots
h2o.partialPlot(object = best_rf, data =
test_h2o, cols = c("A.7", "A.52"))
## boosted trees with H2o
ntrees = c(50, 100, 125); max_depth =
c(4,5,6,7)
learn_rate = c(0.001,0.01, 0.1, 0.5)
learn_rate_annealing = 0.99
gbm_grid <- h2o.grid(algorithm = "gbm",
                     hyper_params =
hyper_params, x = 2:58, y = 59,
                     training_frame = spam_h2o,
seed = 1)
###another good example of trees + plotting
Other good examples of trees
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston) /
2)
Boston.train <- Boston[train, -14]
Boston.test <- Boston[-train, -14]
Y.train <- Boston[train, 14]
Y.test <- Boston[-train, 14]
rf.boston1 <- randomForest(Boston.train, y =
Y.train, xtest = Boston.test, ytest = Y.test,
mtry
                           = ncol(Boston) - 1,
ntree = 500)
rf.boston2 <- randomForest(Boston.train, y =
Y.train, xtest = Boston.test, ytest = Y.test,
mtry
                           = (ncol(Boston) - 1)
/ 2, ntree = 500)
rf.boston3 <- randomForest(Boston.train, y =
Y.train, xtest = Boston.test, ytest = Y.test,
mtry
                           = sqrt(ncol(Boston)
- 1), ntree = 500)
plot(1:500, rf.boston1$test$mse, col = "green",
type = "l", xlab = "Number of Trees", ylab = "T
    est MSE", ylim = c(10, 19))
lines(1:500, rf.boston2$test$mse, col = "red",
type = "l")
lines(1:500, rf.boston3$test$mse, col = "blue",
type = "l")
legend("topright", c("m = p", "m = p/2", "m =
sqrt(p)"), col = c("green", "red", "blue"), cex
=
       1, lty = 1)

###SVM
  library(e1071)
library(kernlab)
## fitting initial model and predicting
svm.model <- svm(class ~ .  , data =
train_data)
svm.pred <- predict(svm.model, test_data[,-
c(length(test_data))])
## test results
svm.model$index
## tuning radial kernel
tune.out.radial <- tune(svm,class ~ . ,data =
train_data, kernal = "radial",  ranges =
list(cost = c(0.1,1,10,100),

gamma = c(0.01, 0.1,0.5,1,2)))
summary(tune.out.radial)
plot(tune.out.radial, main = 'Radial')

### Better SVM plots
dat=data.frame(x,y=as.factor(y))
library(e1071)
svmfit = svm(y~.,data=dat, kernel="linear",
cost=1000,scale=FALSE)
make.grid=function(x,n=75){
  grange=apply(x,2,range)

x1=seq(from=grange[1,1],to=grange[2,1],length=n
)

x2=seq(from=grange[1,2],to=grange[2,2],length=n
)
  expand.grid(X1=x1,X2=x2)
}
xgrid=make.grid(x)
ygrid=predict(svmfit,xgrid)
plot(xgrid,col=c("red","green")[as.numeric(ygri
d)],pch=20,cex=.2)
points(x,col=y+5,pch=19)
points(x[svmfit$index,],pch=5,cex=2) beta =
drop(t(svmfit$coefs)%*%x[svmfit$index,])
beta0=svmfit$rho
plot(xgrid,col=c("red","green")[as.numeric(ygri
d)], pch=20,cex=.2)
points(x,col=y+5,pch=19)
abline(beta0/beta[2],-beta[1]/beta[2])
abline((beta0-1)/beta[2],-
beta[1]/beta[2],lty=2)
abline((beta0+1)/beta[2],-
beta[1]/beta[2],lty=2)

###ROC Curves
  library(ROCR)
pred1 <- prediction(yhatTrain,
split.data$train$spam)
perf1 <- performance(pred1,"tpr","fpr")
plot(perf1, col= 'red')
auc = performance(pred1, measure =
"auc")@y.values[[1]]
print(auc)

###Neural Net:
  library(h2o)
localH2O = h2o.init(ip = "localhost", port =
54321, startH2O = TRUE)
#creating h2o datasets
datTrain_h2o <- as.h2o(datTrain)
## fitting the initial model
model <- h2o.deeplearning(x = 2:11, y = 12,
training_frame = datTrain_h2o, validation_frame
= datTest_h2o, nfolds = 10,
export_weights_and_biases = T, seed = 123)
summary(model)
## plotting the variable importance
h2o.varimp_plot(model)
## Evaluating performance
yhat_test <- h2o.predict(model,
datTest_h2o)$predict
yhat_test <- as.factor(as.matrix(yhat_test))
prprint(table(yhat_test, data_num[-
train,]$class))
## plotting a neural network
library(NeuralNetTools)
wts <- c()
for(l in
1:(length(model@allparameters$hidden)+1)){
  wts_in <- h2o.weights(model, l)
  biases <- as.vector(h2o.biases(model, l))
  for (i in 1:nrow(wts_in)){
    wts <- c(wts,
biases[i],as.vector(wts_in[i,]))}}
struct <- model@model$model_summary$units
plotnet(wts, struct = struct)
## grid search
epochs <- c(5,50,100,150)
activations <- c("Maxout", "Tanh", "Rectifier")
hidden <- list(c(5,5,5,5), c(30,30,30),
c(60,60), c(120))
hyper_params <- list(epochs = epochs,
activation = activations, hidden = hidden)
grid <- h2o.grid(algorithm = "deeplearning",
                 hyper_params = hyper_params, x
= 2:11, y = 12,
                 training_frame = datTrain_h2o,
validation_frame = datVal_h2o)
## best model
grid@summary_table[1,]
best_model <- h2o.getModel(grid@model_ids[[1]])
print(best_model@allparameters)
print(h2o.performance(best_model))
print(h2o.logloss(best_model, valid=T))
h2o.confusionMatrix(best_model, valid = TRUE)
```