-Goal: Practice basic R commands/methods for classification. If you are already familiar with some of the commands/methods, you can just practice the ones that are new to you.

\*> # Blue text are explanations.

\*> Blue text are R commands/methods your will type into an R console.

\*Black and Red text are output from R

**Chapter 8 Classification R Exercise: Decision Trees and Ensemble Methods**

#We will use a number of decision tree and ensemble algorithms, and examine the learning process in this exercise:

#Packages used in this exercise:

# rpart: Closely follows the ideas in the seminal book Classification and Regression Trees by Breiman et al. (1984)

# rpart.plot: rpart plotting support

# DMwR2: Functions developed to support exercises the book Data Mining with R 2nd Ed.

# rsample: support data splitting

# dplyr: data wrangling

# adabag: bagging and boosting

# ipred: another bagging

# randomForest: randomForest

# gbm: gradient boosting, another boosting

**> #Build and visualize single decision tree using rpart**

**> #classification**

> # Package rpart provides two main functions to assist in obtaining tree-based models: (i) one for growing the trees (named rpart()); and (ii) the other for post-pruning them (named prune.rpart()). Our book package provides function rpartXse() that joins the two steps in a single function call, which is more practical in most cases. This function calls the functions implemented in package rpart to obtain trees that are post-pruned using the X-SE rule. Trees in rpart are grown till one of three criteria is true: (1) the decrease in

the error of the current node goes below a certain threshold; (2) the number of samples in the node is less than another threshold; or (3) the tree depth exceeds another value. These thresholds are controlled by the parameters cp, minsplit, and maxdepth, respectively.

> # Given a tree obtained with the rpart() function, R can produce a set of

subtrees of this tree and estimate their predictive performance using a procedure based on cross validation estimates. For each subtree an error estimate is obtained together with a standard error of this estimate. The X-SE pruning rule determines that the final selected tree is the smallest tree in the sequence whose estimated error is smaller than the lowest error plus the respective standard error. For instance, if the tree in the sequence of subtrees with the lowest error has an estimate of 4.5±0.4 and has 13 nodes, but there is

a (smaller) tree in the sequence with 10 nodes and an estimated error of 4.8, this latter tree will be selected if we are using the 1-SE rule because 4.8 < 4.5 + 0.4 and this rule prefers smaller trees. Function rpartXse() allows you to specify the number of standard errors to use in the X-SE pruning rule, i.e the value of X. It then grows a very large tree using very relaxed values of parameters cp and minsplit, which then is pruned using the procedure

described above.

> library(DMwR2)

> set.seed(1234)

> data(iris)

> #build a tree: use all attributes in iris dataset to predict Species name.

> #function arguments explained in the comments before: se, minsplit, maxdepth and cp.

> ct1 <- rpartXse(Species ~ ., iris, se=1, cp=0, minsplit = 6, maxdepth = 10)

> #build a tree using 0-SE pruning (select the lowest estimated error subtree of the original overly large tree)

> ct2 <- rpartXse(Species ~ ., iris, se=0, cp=0, minsplit = 6, maxdepth = 10)

> #plot the trees

> install.packages("rpart.plot")

> library(rpart.plot)

> #the tree can be displayed with a variety of features controlled by the parameters supplied to prp()

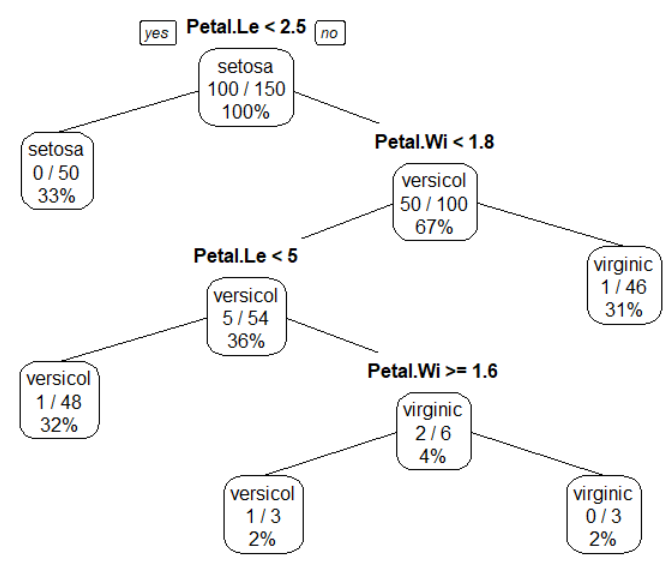
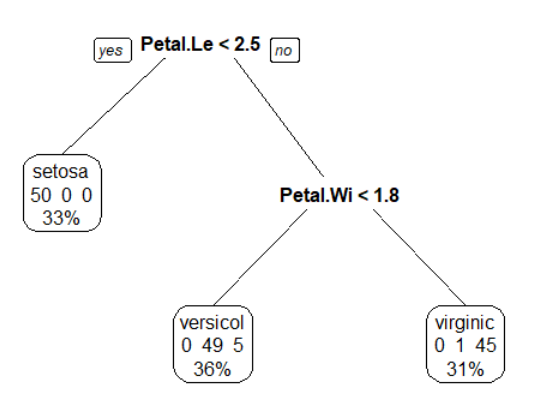
> #e.g: type=0, only displays labels at terminal nodes, type=1 also for internal nodes

> #extra=1 Display the number of observations that fall in the node (per class for class objects;)

> #extra=3 Class models: misclassification rate at the node, expressed as the number of incorrect classifications and the number of observations in the node.

> #extra=+100  Add 100 to any of the above to also display the percentage of observations in the node.

> prp(ct1, type=0, extra=101, roundint = FALSE)



> prp(ct2, type=1, extra=103, roundint = FALSE)

**> #now we know how to produce a decision tree for classification**

**> #let’s do it again with training and test data**

>

> set.seed(1234)

> nrow(iris)

[1] 150

> #100 training and 50 test

> rndSample <- sample(1:nrow(iris),100)

> #training examples

> tr <- iris[rndSample, ]

> #rest is the test examples

> ts <- iris[-rndSample, ]

> #build a tree

> ct <- rpartXse(Species ~ ., tr, se=0.5)

> #use ct to predict class labels for the test examples

> ps1 <- predict(ct, ts) #this gives the probably of an instance belong to a class

> head(ps1)

setosa versicolor virginica

1 1 0 0

3 1 0 0

9 1 0 0

11 1 0 0

12 1 0 0

14 1 0 0

> ps2 <- predict(ct, ts, type="class") #this gives the class label

> head(ps2)

1 3 9 11 12 14

setosa setosa setosa setosa setosa setosa

Levels: setosa versicolor virginica

> #now lets create a contingency table and see how well the classifier works on test examples

> #compare ps2 results (machine predication) with correct labels in test

> (cm <- table(ps2, ts$Species))

ps2 setosa versicolor virginica

setosa 12 0 0

versicolor 0 21 3

virginica 0 0 14

> #find the error rate in percentage

> (1-sum(diag(cm))/sum(cm))

[1] 0.06

**> For numerical prediction, train a regression tree**

**> Use rpart to learn a regression tree**

> #regression tree: based on http://uc-r.github.io/regression\_trees

> install.packages("rsample")

> install.packages("ipred")

> install.packages("AmesHousing") #dataset

> library(rsample) # data splitting

> library(dplyr) # data wrangling

> library(rpart) # performing regression trees>

> library(rpart.plot) # plotting regression trees

> library(ipred) # bagging

> # Create training (70%) and test (30%) sets for the AmesHousing::make\_ames() data.

> # Use set.seed for reproducibility

>

> set.seed(123)

> ames\_split <- initial\_split(AmesHousing::make\_ames(), prop = .7)

> ames\_train <- training(ames\_split)

> ames\_test <- testing(ames\_split)

> #examine the structure of ames\_train

> str(ames\_train)

Classes ‘tbl\_df’, ‘tbl’ and 'data.frame': 2051 obs. of 81 variables:

$ MS\_SubClass : Factor w/ 16 levels "One\_Story\_1946\_and\_Newer\_All\_Styles",..: 1 1 6 6 12 12 6 6 1 6 ...

> #learn a model using rpart

> m1 <- rpart(

+ formula = Sale\_Price ~ .,

+ data = ames\_train,

+ cp = 0.01,

+ method = "anova"

+ )

>”anova” is splitting method for regression. Its splitting criteria is SST − (SSL + SSR), where SST = Sum((yi − y¯)^2) is the sum of squares for the node, and SSR, SSL are the sums of squares for the right and left son, respectively. For classification, method=”class”.

> m1 # format of a decision node: “node), split, n, deviance, yval”

n= 2051

**node), split, n, deviance, yval**

**\* denotes terminal node**

1) root 2051 1.329920e+13 181620.20

2) Overall\_Qual=Very\_Poor,Poor,Fair,Below\_Average,Average,Above\_Average,Good 1699 4.001092e+12 156147.10

4) Neighborhood=North\_Ames,Old\_Town,Edwards,Sawyer,Mitchell,Brookside,Iowa\_DOT\_and\_Rail\_Road,South\_and\_West\_of\_Iowa\_State\_University,Meadow\_Village,Briardale,Northpark\_Villa,Blueste 1000 1.298629e+12 131787.90

8) Overall\_Qual=Very\_Poor,Poor,Fair,Below\_Average 195 1.733699e+11 98238.33 \*

9) Overall\_Qual=Average,Above\_Average,Good 805 8.526051e+11 139914.80

18) First\_Flr\_SF< 1150.5 553 3.023384e+11 129936.80 \*

19) First\_Flr\_SF>=1150.5 252 3.743907e+11 161810.90 \*

5) Neighborhood=College\_Creek,Somerset,Northridge\_Heights,Gilbert,Northwest\_Ames,Sawyer\_West,Crawford,Timberland,Northridge,Stone\_Brook,Clear\_Creek,Bloomington\_Heights,Veenker,Green\_Hills 699 1.260199e+12 190995.90

10) Gr\_Liv\_Area< 1477.5 300 2.472611e+11 164045.20 \*

11) Gr\_Liv\_Area>=1477.5 399 6.311990e+11 211259.60

22) Total\_Bsmt\_SF< 1004.5 232 1.640427e+11 192946.30 \*

23) Total\_Bsmt\_SF>=1004.5 167 2.812570e+11 236700.80 \*

3) Overall\_Qual=Very\_Good,Excellent,Very\_Excellent 352 2.874510e+12 304571.10

6) Overall\_Qual=Very\_Good 254 8.855113e+11 273369.50

12) Gr\_Liv\_Area< 1959.5 155 3.256677e+11 247662.30 \*

13) Gr\_Liv\_Area>=1959.5 99 2.970338e+11 313618.30 \*

7) Overall\_Qual=Excellent,Very\_Excellent 98 1.100817e+12 385440.30

14) Gr\_Liv\_Area< 1990 42 7.880164e+10 325358.30 \*

15) Gr\_Liv\_Area>=1990 56 7.566917e+11 430501.80

30) Neighborhood=College\_Creek,Edwards,Timberland,Veenker 8 1.153051e+11 281887.50 \*

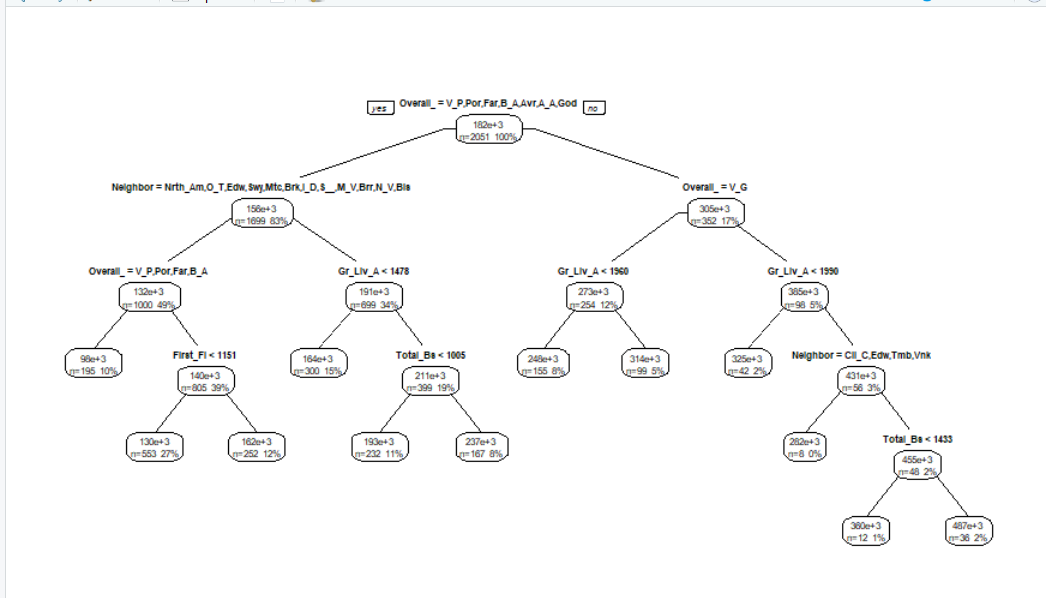
31) Neighborhood=Old\_Town,Somerset,Northridge\_Heights,Northridge,Stone\_Brook 48 4.352486e+11 455270.80

62) Total\_Bsmt\_SF< 1433 12 3.143066e+10 360094.20 \*

63) Total\_Bsmt\_SF>=1433 36 2.588806e+11 486996.40 \*

> #visualize the tree

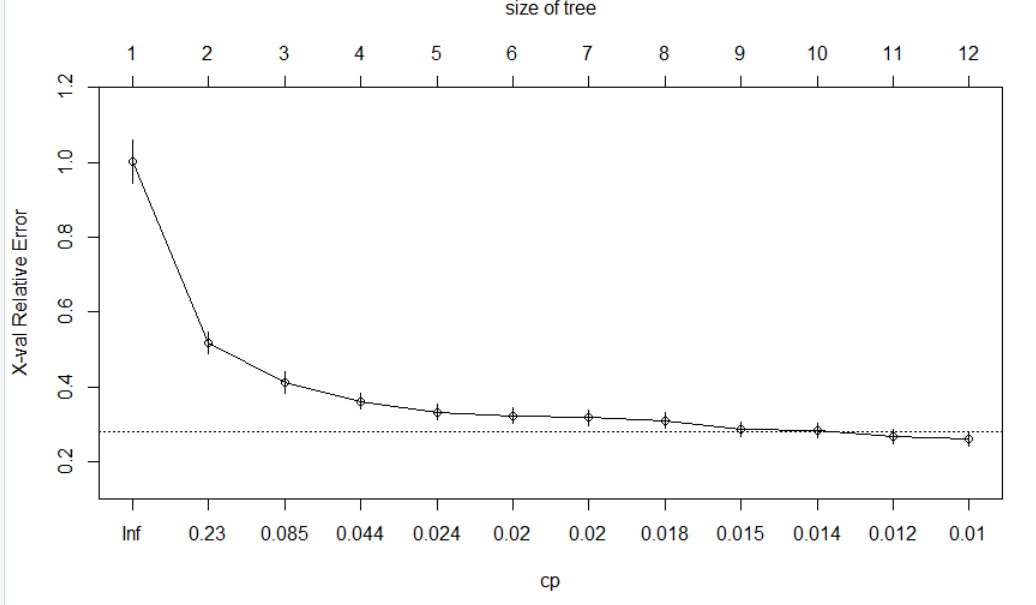
> prp(m1, type=1, extra=101, roundint = FALSE)



>

**> #Peek inside the learning process: use plotcp() function**

> plotcp(m1)



> #In the plot, y-axis is cross validation error, lower x-axis is cost complexity (α or cp) value, upper x-axis is the number of terminal nodes (tree size)

> #We find diminishing returns after 12 terminal nodes.

> #You may also notice the dashed line which goes through the point tree size= 9.

> #Breiman et al. (1984) suggested that in actual practice, its common to instead use the smallest tree within 1 standard deviation of the minimum cross validation error (aka the 1-SE rule). Thus, we could use a tree with 9 terminal nodes and reasonably expect to experience similar results within a small margin of error.

> #We can also examine the effects of pruning by generating a full tree (cp=0).

>

> m2 <- rpart(

+ formula = Sale\_Price ~ .,

+ data = ames\_train,

+ method = "anova",

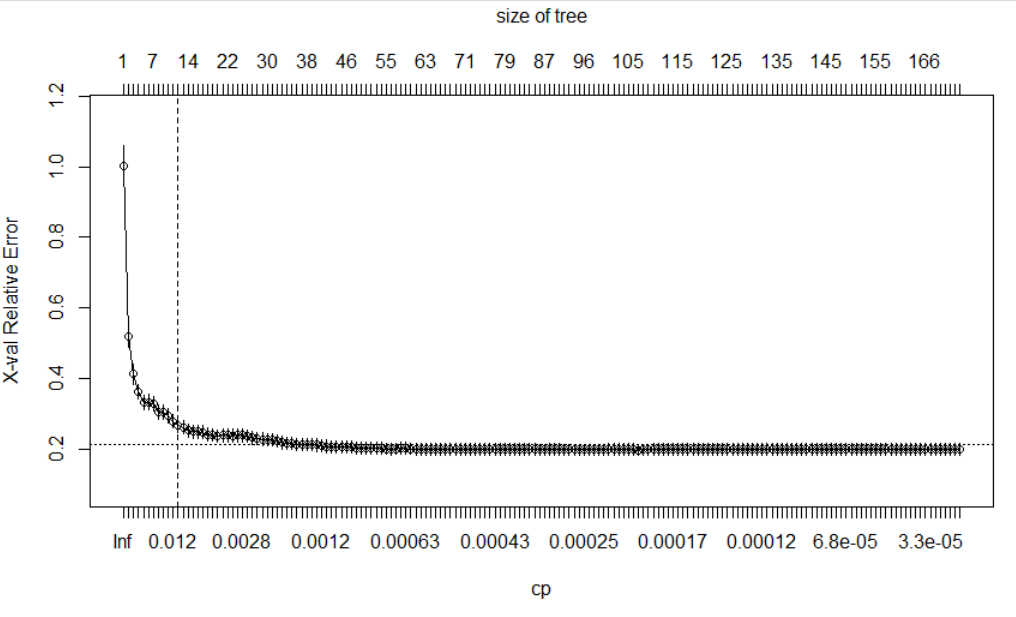
+ control = list(cp = 0, xval = 10)

+ )

>

> plotcp(m2)

> abline(v = 12, lty = "dashed") #vertical line at tree size = 12.



>In the plot, We see a much larger tree with over 166 terminal nodes, although cp reduces (neglect able amount), cross=validation error stopped reducing around treesize=12. All those additional splitting were over-fitting.

> #To compare the error for each α value, rpart performs a 10-fold cross validation so that the error associated with a given α value is computed on the hold-out validation data.

> #try to improve on the m1 model

> m1$cptable

CP nsplit rel error xerror xstd

1 0.48300624 0 1.0000000 1.0017486 0.05769371

2 0.10844747 1 0.5169938 0.5189120 0.02898242

3 0.06678458 2 0.4085463 0.4126655 0.02832854

4 0.02870391 3 0.3417617 0.3608270 0.02123062

5 0.02050153 4 0.3130578 0.3325157 0.02091087

6 0.01995037 5 0.2925563 0.3228913 0.02127370

7 0.01976132 6 0.2726059 0.3175645 0.02115401

8 0.01550003 7 0.2528446 0.3096765 0.02117779

9 0.01397824 8 0.2373446 0.2857729 0.01902451

10 0.01322455 9 0.2233663 0.2833382 0.01936841

11 0.01089820 10 0.2101418 0.2687777 0.01917474

12 0.01000000 11 0.1992436 0.2621273 0.01957837

> #tune the model: try out different combinations of minsplit and maxdepth values and compare the models

> hyper\_grid <- expand.grid(

+ minsplit = seq(5, 20, 1),

+ maxdepth = seq(8, 15, 1)

+ )

> head(hyper\_grid)

minsplit maxdepth

1 5 8

2 6 8

3 7 8

4 8 8

5 9 8

6 10 8

> #total number of combinations (trees to be generated)

> nrow(hyper\_grid)

[1] 128

> #use a for-loop to run the combinations one by one and save the trees/models.

> models <- list()

> for (i in 1:nrow(hyper\_grid)) {

+

+ # get minsplit, maxdepth values at row i

+ minsplit <- hyper\_grid$minsplit[i]

+ maxdepth <- hyper\_grid$maxdepth[i]

+

+ # train a model and store in the list

+ models[[i]] <- rpart(

+ formula = Sale\_Price ~ .,

+ data = ames\_train,

+ method = "anova",

+ control = list(minsplit = minsplit, maxdepth = maxdepth)

+ )

+ }

> # function to get optimal cp

> get\_cp <- function(x) {

+ min <- which.min(x$cptable[, "xerror"])

+ cp <- x$cptable[min, "CP"]

+ }

> # function to get minimum error

> get\_min\_error <- function(x) {

+ min <- which.min(x$cptable[, "xerror"])

+ xerror <- x$cptable[min, "xerror"]

+ }

> hyper\_grid %>%

+ mutate(

+ cp = purrr::map\_dbl(models, get\_cp),

+ error = purrr::map\_dbl(models, get\_min\_error)

+ ) %>%

+ arrange(error) %>%

+ top\_n(-5, wt = error)

minsplit maxdepth cp error

1 6 13 0.0108982 0.2421256

2 7 8 0.0100000 0.2453631

3 13 10 0.0100000 0.2454067

4 9 13 0.0100000 0.2459588

5 20 9 0.0100000 0.2460173

> #use the parameter combination that results in the least error to build the final tree

> #because there are additional random factors in the tree building process, your results may not be exactly the same as mine. cp=0.01 was the default values used for all trees

> optimal\_tree <- rpart(

+ formula = Sale\_Price ~ .,

+ data = ames\_train,

+ method = "anova",

+ control = list(minsplit = 7, maxdepth = 8, cp = 0.01)

+ )

> pred <- predict(optimal\_tree, newdata = ames\_test)

> pred

1 2 3 4 5 6 7

129936.78 161810.91 247662.29 313618.26 247662.29 236700.82 164045.16

8 9 10 11 12 13 14

129936.78 98238.33 129936.78 129936.78 325358.26 236700.82 236700.82

15 16 17 18 19 20 21

164045.16 192946.28 164045.16 247662.29 236700.82 192946.28 164045.16

22 23 24 25 26 27 28

192946.28 192946.28 129936.78 161810.91 192946.28 192946.28 164045.16

29 30 31 32 33 34 35

236700.82 164045.16 247662.29 129936.78 236700.82 236700.82 161810.91

36 37 38 39 40 41 42

129936.78 129936.78 129936.78 129936.78 161810.91 161810.91 161810.91

43 44 45 46 47 48 49

…

> #model error: Root Mean Squared Error. on average, our predicted sales prices are about $39,145 off from the actual sales price.

> RMSE(pred = pred, obs = ames\_test$Sale\_Price)

[1] 39145.39

|  |
| --- |
| > #Single tree models suffer from high variance: different random samples will likely produce different trees  > #We see the prediction performance is not very good even after tuning.  > #Model ensembles are a way to reduce performance variations, for decision trees, random forest is a robust approach to improve tree prediction performance  **> #Ensemble methods for classification and regression**  > set.seed(1234)  > rndSample <- sample(1:nrow(iris),100)  > tr <- iris[rndSample, ]  > ts <- iris[-rndSample, ]  **> #AdaBoost: offers bagging and boosting, both are for classification (not regression)**  > install.packages("adabag")  > library(adabag)  > #bagging: learn trees on boostrapped samples using all variables  > ct <- bagging(Species ~ ., tr, mfinal=500)  > ps2 <- predict(ct, ts)  > ps2$confusion  Observed Class  Predicted Class setosa versicolor virginica  setosa 12 0 0  versicolor 0 20 1  virginica 0 1 16  > ps2$error #reduced from 0.06 to 0.04  [1] 0.04  > #boosting:iteratively add new models to the ensemble, each model tries to overcome the errors made by the previous model  >  > ct <- boosting(Species ~ ., tr, mfinal=500)  > ps2 <- predict(ct, ts)  > ps2$confusion  Observed Class  Predicted Class setosa versicolor virginica  setosa 12 0 0  versicolor 0 20 1  virginica 0 1 16  > ps2$error  [1] 0.04  **> #Another algorithm: Bagging, can do regression**  **> #bagging: learn trees on boostrapped samples using all variables**  > #750 trees built to bootstrapped samples using all variables  > rfm <- ipred::bagging(Sale\_Price ~ ., ames\_train, nbag=750)  > rfpred <- predict(rfm, ames\_test)  > #error reduced from 39145 to 34536  > RMSE(rfpred, ames\_test$Sale\_Price)  [1] 34536.11 |
| **> #Another algorithm: RandomForest, classification and regression** |
| |  | | --- | | > install.packages(“randomForest”)  > library(randomForest)  > #build model based on a forest of 750 trees  > rfm <- randomForest(Sale\_Price ~ ., ames\_train, ntree=750)  > #predict test data  >rfpred <- predict(rfm, ames\_test)  > #error reduced from 39145.39 to 24226.48.  > RMSE(rfpred, ames\_test$Sale\_Price)  [1] 24226.48  > **#Another algorithm: Gradient boosting:** classification and regression. like AdaBoost, but use gradient descent to address the current errors. Can overfit, so using cross-validation is very important (cv.folds)  > install.packages("gbm")  > library(gbm)  > rfm <- gbm(Sale\_Price ~ ., data=ames\_train, n.trees=750, cv.folds=5)  Distribution not specified, assuming gaussian ...  > rfpred <- predict(rfm, ames\_test, n.trees=750)  > RMSE(rfpred, ames\_test$Sale\_Price)  [1] 26995.43  I didn’t include a Naïve Bayes exercise here, but if you are interested, check out <https://www.r-bloggers.com/understanding-naive-bayes-classifier-using-r/>.  **[REQUIRED]**  Exercises on your data set:   1. Build a decision tree (regression or classification) and use one ensemble method on your own dataset. 2. List at least two interesting rules from the decision tree your generated. 3. Does the ensemble method improve the performance on your dataset? | |