THe tree-based method

Statistical Analysis using the CART method (classification and regression trees)

[A picture containing chart

Description automatically generated](https://uc-r.github.io/regression_trees)[](https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/)

17 December 2021

Baruch College

CIS 3920

- By Muhammad Megahed

**Table of Content**

[Part A – Introduction to stock data 2](#_Toc90597904)

[A1 – Working through the tree-based method. 2](#_Toc90597905)

[A2 – Challenging aspect of the tree-based method work through. 8](#_Toc90597906)

[A3 – Creating a classification plot. 9](#_Toc90597907)

[Part B 14](#_Toc90597908)

[(1) 14](#_Toc90597909)

[(KNN) 14](#_Toc90597910)

[(Naïve Bayes) 15](#_Toc90597911)

[(GLM) 16](#_Toc90597912)

[(Tree) 17](#_Toc90597913)

[(2) – Applying classification methods on the last 500 days of the stock. 18](#_Toc90597914)

[(KNN) 18](#_Toc90597915)

[(Naïve Bayes) 18](#_Toc90597916)

[(GLM) 19](#_Toc90597917)

[(Tree) 19](#_Toc90597918)

[Table of comparison: 21](#_Toc90597919)

[Appendix: 22](#_Toc90597920)

# Part A – Introduction to stock data

## A1 – Working through the tree-based method.

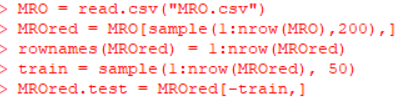
My stock name is Marathon Oil Corporation which has a ticker name of MRO. I have the history data of the stock saved in a csv file under the name “MRO.csv”. The full stock data is a little over 3900 rows because it’s from January 2006 until September 2021. It’s too big. So, in this project I will draw a sample of 200 rows only and I will call it my “reduced-size stock risk data set” or sometimes just “MRO data set”. So, whenever I say MRO data set then know that I mean the sample of the 200 rows, which I will show how I generate as well. Before I start this is a general look of the data set:  
Table

Description automatically generated

As you can see the data set has a few rows. The most significant ones are the Risk column, and the LRLag1 and LRLag2 which are the standardized logged lag 1 and 2 of the stock range data. I will be using the Tree-Based Methods (or *CART*, short for *classification and regression trees*) which is a classification methodology, and I will be using it on my reduced-size data set.

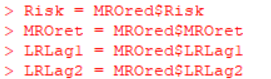
* Fitting a classification tree:

First things first, I loaded MRO.csv into R. Then, I proceeded to draw the sample of 200 rows. First I created a variable using the sample() function in R assigning it to take 200 random values out of whatever how many rows MRO has. I called that variable MROred.



As you may have noticed I also used the command nrow() to reset the index number of the new set from 1:200 after they were picked randomly from the original dataset.

I then created different variables for the columns that I think matter the most in the MROred:



Our Risk column consists of 1s and 0s which determine whether the stock is risky (1) or not (0) based on the range of the stock that day.

First, I will use the tree() function to create a classification tree that predicts the values of Risk using all the other variables. This goes like the following:



To show how that went we use the summary() function to display a general look on our classification tree:

Text

Description automatically generatedFrom the look of it we notice that the terminal nodes or what you may call the “leafs” or the “end decisions” that we have are only 2. This means that this is a very short tree and that can be shown as well by plotting the tree. You can also notice that our little tree used one variable only which is Range. This makes perfect sense since Range is the variable we used to create the Risk column. To show you a quick look of the tree we can use the plot() function:

A picture containing square

Description automatically generated

Basically the plot is telling us that the tree had only one Decision Node (split node) and that node is also the root node. The Range variable is the one that was used because the condition to creating the Risk column was that “If Range > 5.8, then choose ‘1’, if not then choose ‘0’” and that is shown perfectly in that tree plot. Now let’s go ahead and try to recreate that tree without using the Range variable.

I rewrote the tree() function while adding “-Range” to it so it excludes it. Then I ran the summary function again:

Text

Description automatically generated

Ok! This looks a lot more promising here!

Now I will use the tree() function again to do the same thing but I will use a train and a test set. First, I created a train variable with 50 rows that I will use as the train set, and I also created a test set. That way I could build the tree using the training set, and evaluate its performance on the test data using the predict() function. I also used the argument “type = ‘class’” to make R return the actual class predistortion:



Graphical user interface, text, application

Description automatically generated

As seen, the table created from the predict function shows that the approach leads to correct predictions of 44 of the 0 values and 61 of the 1 values. To calculate the prediction rate, we do simple math as follows:



This means that we have a prediction rate of 70%.

Now we have an option to consider which is called “pruning” the tree which is basically like cross-validation. Hence the name of the function we use is cv.tree(). Now this function uses “deviance” to guide the pruning process, that means that it uses the difference between the observed value of a variable and its mean, but we don’t want that. That’s why we will use the “FUN=prune.misclass” argument in our function which forces it to use classification error-rate to guide the class validation and pruning process:

Text

Description automatically generatedThe “size” here corresponds to the number of terminal nodes on each tree considered. The “dev” is the error rate associated with each tree. The “k” is the cost-complexity parameter. Now the k part is confusing to me as much as it may be confusing to you. “k” is the minimum improvement in the model needed at each node or it can also be defined as the tuning parameter α for cost-complexity pruning. But why is it set to -infinity. This is something that I will be looking into later. What matters now in the summary that we have is that we realize that a tree with 2 terminal nodes has the lowest number of cross-validation errors which is 16.

Now let’s try and test how the pruned tree (the one with 2 terminal nods) would perform in the classification process:

A picture containing shape

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

The accuracy level in this case went down to 68.67% and the only variable used in the construction process was RangeLag2. So, pruning the tree didn’t help us with getting better results which isn’t always the case. I also tested the other trees by changing the value in the argument “pretty = 2” to the other values, but I still received less than 70% in each case.

* Fitting a Regression tree:

The process of fitting a regression tree is almost the same as fitting a classification tree. Except that we use the mean as a deviance instead of using the classification error like we did before. We also use an extra library called MASS which is basically Modern Applied Statistics with S.

This is how I installed the MASS package and ran the tree again:

Text

Description automatically generated

A picture containing text

Description automatically generated

Diagram, timeline

Description automatically generated

Notice that I also changed the train values from 50 to 100. If we wish to prune the tree we use the prune.tree() function as we did before. But for now, we will use the unpruned tree to make predictions on the test set.

Text

Description automatically generated

The 0.224 value I calculated is the MSE or the Mean Square Error. By taking the square root of that value, you get a value of 0.47 which basically mean that this model leads to test predictions that are within around 0.47 of the stock risk value.

I personally believe that predicting Risk using this model isn’t really a great idea. Fitting a regression tree should be used on a variable that has a wide variance so the “mean” would mean something. Like using it on the stock Range or as in the ISLR book it was used on a Median Home Value of the suburbs in a data set called “Boston”.

* Bagging:

Bagging is basically using each predictor (Variable) in every single split of the tree. In this attempt we will use a library called “randomForest” which helps us use the bagging method by applying an argument called “mtry”:

Graphical user interface, text, application, email

Description automatically generated

Now let’s test how well it performed compared to the optimally pruned single tree we did before:

A picture containing company name

Description automatically generated

This is very disappointing. The value received here should be lower than 0.224 for it to be considered an improvement in the prediction process. I guess what I talked about with fitting a regression tree also applies to the bagging process with randomForest.

What if we try it with a lower value of mtry? It’s known that the default for randomForest() function is to use a value of p/3 in a regression tree, and sqrt(p) when building a classification tree. This time we will use 4 for mtry:

A picture containing text

Description automatically generated

As expected, it was slightly lower but nothing significant. Let’s see how important was each variable in this process by using the importance() function:

This shows that Open (stock open price) and High (stock highest price) were the most important variables in the prediction process.

## A2 – Challenging aspect of the tree-based method work through.

Previously when we were doing the cross-validation for our classification tree, I typed “cv.MROred” which gave me the following:

We said that the “size” here corresponds to the number of terminal nodes on each tree considered. The “dev” is the error rate associated with each tree.Text

Description automatically generated We also said that the “k”, which is the cost-complexity parameter, is the minimum improvement in the model needed at each node or it can also be defined as the tuning parameter α for cost-complexity pruning. But why is it set to negative infinity for the first tree?!

I tried doing some investigation about this part. First, I had to find a deeper explanation for what k or α is to begin with.

I came across a few helpful websites with users trying to put an explanation in English rather than just pure statistics talk. I can’t call myself great at statistics, in fact it’s one of my least favorite subjects.

In [this](https://stats.stackexchange.com/questions/117908/rpart-complexity-parameter-confusion) website for example, the user was using cross-validation on a tree as well, but his problem was a little different. He expected the first node to have a k value of 0.4 but instead he received a 0.5. Another user in the comments explained it as follows: “the complexity parameter α specifies how the cost of a tree C(T) is penalized by the number of terminal nodes |T|, resulting in a regularized cost Cα(T)”.

I still had some trouble understanding what it means. The user went on to say that “Small α results in larger trees and potential overfitting, large α in small trees and potential underfitting”. One other user said that the cost-complexity parameter isn’t the error in the particular node, it’s the amount by which splitting that node improved the relative error.

If I was to understand that I can look again at that picture and notice how k moved to being negative infinity from 0.00000 when the tree split from having 6 nodes to 8 nodes, and it also went from 1.00000 to 0.00000 when it split from 5 to 6 nodes. I think that means that splitting the nodes in our case is improving the relative error by the negative value…which basically means that it’s having a reverse effect in our case.

This makes sense because if you recall when we tried using different node values in cross-validation we got less than 70% in all of them compared to the one with 2 nodes.

## A3 – Creating a classification plot.

So, in A1 in the fitting of a classification tree part, we have a tree with 13 terminal nodes and apparently 8 variables were used in creating that tree. Now let’s see how we can create a plot of that. The easiest method is using the plot() function then using the text() function to add info to the plot:



Timeline

Description automatically generated

I also created the cross-validation function cv.MROred and created a summary function with “k”, “dev” and “size”. let’s plot this error rate as a function using “size” once and “k” once:

A picture containing text

Description automatically generated

Chart, line chart

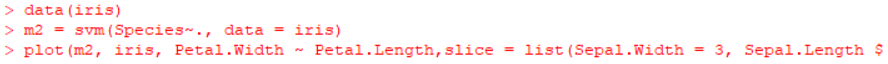
Description automatically generated

So, as seen it was easy to plot the created classification trees and other dependent plots. The hard part was to find a way to plot a classification space plot. After researching I realized that SVM is the most (perhaps the only) methodology used in creating classification space plots.

It also came to my attention that the svm() function is part of a package called e1071 which I read the documentation the best I could.

I tried using svm() to create a classification space knowing that it was not my chosen method. However, I have not been able to reach any further than an error and a plot command that runs without actually plotting anything. I even downloaded and installed RStudio to see if the R console is at fault here but it also had the same outcome

I even went ahead and tried an example that was used on the iris data successfully as shown below:



Chart

Description automatically generated

Now after attempting that on my dataset, I first get an error in the data command that says that MROred is not found. I tested the dataset with the class() function and it clearly shows that it indeed does exist:

Logo

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidenceSo I thought maybe Iris has a different class that’s why it’s run through the data() function with no errors. But I was disappointed again to find that it isn’t special at all:

So this got me thinking that maybe the data command isn’t really necessary. After all the svm() function has an argument where you enter your data after all. Again…nothing but disappointment:

Text

Description automatically generated with low confidence

I then tried running the models raw in the console to see if there was an actual success of any kind in my case:

Graphical user interface

Description automatically generated with low confidenceI immediately noticed the SVM-Type which showed a c-classification for the iris data and eps-regression for my data.

That got me thinking that this might be why the plot is not coming up!

My next mission now is to try and figure out how to change the SVM-Type.

I checked the [documentation for the svm](https://www.rdocumentation.org/packages/e1071/versions/1.7-9/topics/svm) function again to try to see if type is in-fact an argument, and sure enough I was correct!

Now this is the moment of truth…but first let me review my function again.



- As seen here I actually used only two variables because otherwise I assume I would have had an unpleasant looking plot. So, my two variables of choice are the LRLag1 and LRLag2.

- My data is MROred which didn’t require the data() function in the first place and it cost me a lot of time to figure that out.

- Type is set to “C-classification” to ensure that I will get the plot to actually show up.

All looks good now…Let’s plot!



Chart, scatter chart

Description automatically generated

Voila!! Now I have a classification plot of the Risk column using the 2 main variables that I picked, LRLag1 and LRLag2.

Of course, the title there should say “SVM classification plot” but it overlapped with the command I ran to add my name to the graph.

# Part B

(1) – Applying classification methods on a random sample of 500 days.

In this part I will redraw a random sample of 500 days from my stock data, and I will apply multiple methods to them then do a comparison.

First method is called K-Nearest-Neighbor or KNN in short. This function is a non-parametric classification method used for both regression and classification, and in both cases the input consists of the k closest training examples in the data set.

Diagram, schematic

Description automatically generatedA quick example would be the following picture:

The red circle is the one we’re trying to predict. So, in the case where the k=3, we are predicting using the nearest 3 neighbors. That means that we check the nearest 3 colored circles. In this case we can see among the 3 circles around the red one, the blue has 2, so, the red one must be a blue one as well. Same thing applies with the k=6 area. There is 4 out of 6 circles with the color yellow(?), which means the red one should be yellow as well.

Second method will be the naïve Bayes one. This one is a little too long to explain but you can read about it in [here](https://scikit-learn.org/stable/modules/naive_bayes.html).

Third method is Logistic Regression using a function called “glm” which is a generalized linear model and basically it is a flexible generalization of ordinary linear regression. You can also read more about it in [here.](https://www.statmethods.net/advstats/glm.html)

### (KNN)

Ok! Let’s start by drawing the sample of 500. It will be quite similar to what we did before when we drew a sample of 200. So, I will only show the R entries:

First, I created a random sample of 500:

Text

Description automatically generated

I decided to make my train set to have 30% of the values and for the test set to be 70%. So, my train set will have 150 values and my test set will have 350.

Also, when creating my train and test sets I will be removing the risk column and only using LRLag1 and LRLag2 as my train set. I will use the cbind function for that purpose.

The KNN function has a third argument besides the train and set ones, the argument is called “cl” and it’s basically the factor of true classifications of training set. This is why it will be the train set of the Risk column. We will also create a Risk.test variable which we will use to compare our predictions too at the end.

Text

Description automatically generated

Now to run the knn function which is part of the “class” packages. That’s why we have to run “library(class)” first or we will get a “function not found” error.

A picture containing company name

Description automatically generated

Notice that I used k=3 in this case.

We can calculate the prediction rate by doing the following calculation:

A picture containing shape

Description automatically generatedThis means we have a prediction rate of **66.57%**

### (Naïve Bayes)

Now, before we try the second method, Naïve Bayes is based on the Bayes theorem which gives the conditional probability of an event A given another event B has occurred. So, in this part we will split the data set into what you can call 2 events, A and B, or train and test!

If my understanding of the method is correct, the objective is to predict the probability of success in how y.test is predicted using the x.test given the event of x.train predicting y.train.

First, some libraries like “e1071”, “caret”, and “klaR” are required. So we begin by typing the following commands to install them and initialize them:

Text

Description automatically generated

Then, we start creating our variables. Out of the 500 rows I have, I set the first 300 as my train set and the last 200 as my test set. I used the head() and tail() function to do that. Then for each set I created an x and a y variable. The x being the x-variables and the y being the y-variables.

In this part the x-variables are going to be the LRLag1 and LRLag2 which are the columns 16 and 17 in my data set, while the y-variable will be the Risk column which is the 20th column.

Text

Description automatically generated

Then I create the classifier variable which contains the naiveBayes function. The function will have the x.train as the x variable and the y.train as its y.variable:



Now we run the predict function to see how the classifier do with the test set:

A picture containing text

Description automatically generated

This suggests that the predictor had a success rate of **72.5%**

calculations can be done as follows: 

### (GLM)

In this method we will try to estimate a logistic regression model using the glm (generalized linear model) function.

First, we convert Risk to a factor to indicate that rank should be treated as a categorical variable.

A picture containing shape

Description automatically generated

We then install the needed library and run it:



Then we create the glm.fit model using the glm function. In this function as well I will use only LRLag1 and LRLag2 as my predictors, and as usual we will be predicting the Risk column:



Then I use the predict function to test the efficiency of the model:



And then I round the predicted variables to make it either 0 or 1 and then I create the table to calculate the success rate:

Text

Description automatically generated

This gives us a prediction rate of **74%.**

### (Tree)

Now I will attempt to fit a classification tree using the tree method again but for our 500 rows sample. I will post the code only since I already explained what I was doing in part A. This time however I will only use the LRLag1 and LRLag2 columns as my predictors just like I did with the 3 methods before. I will also use 200 rows for the train set and 300 for the test set:

Text

Description automatically generated with low confidence



This gives us a prediction rate of **64%.**

## (2) – Applying classification methods on the last 500 days of the stock.

In this part I will attempt the same 4 methods again, but I made some changes to my data set. I will only be working with the last 500 rows of the data set which are the most recent 500 days.

I also redefined the Risk column to be based on the median of those 500 rows.

### (KNN)

Text, letter

Description automatically generated

Prediction rate: **61.1%**

### (Naïve Bayes)

Text, letter

Description automatically generated

Prediction rate: **58%.**

### (GLM)

Text, letter

Description automatically generated

Prediction rate: **68.2%**

### (Tree)

Text

Description automatically generated

Prediction rate: **63.6%**

As you may know the last 500 days had a higher risk than usual which explains why we have a drop in the prediction rate of all the methods.

In the last 500 days the Risk was very high based on the Range which had a relatively low median. The stock went up and down in a very aggressive way. This can be noticed from the stock data around the first few months of 2020 until 2020. The pandemic had a huge effect on the stock especially knowing that the stock is for an oil company. We notice that the stock started going back up around the beginning of 2021. This makes the last 500 days very unstable because of outside events which will produce low prediction results.

Table

Description automatically generatedTable

Description automatically generated

### Table of comparison:

Here is an overview of how it looks in comparison:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sample\Method** | **KNN** | **Naïve Bayes** | **GLM** | **Tree** |
| **Random 500 days** | 66.57% | 72.5% | 74% | 64% |
| **Last 500 days** | 61.1% | 58% | 68.2% | 63.6% |

# Appendix:

1. Fitting Classification Trees

* Downloading ISLR and loading it:   
    
  
* Using classification trees to analyze the Carseats data set and creating High variable using the ifelse() function on the Sales column:  
    
  
* Using data.frame() to merge High with the rest of the dataset:  
  
* Installing and using the tree() function to fit the a classification tree in order to predict High using all variables but Sales:  
  Text

  Description automatically generated  
  A picture containing text

  Description automatically generated
* Using summary() function to list the variables that were used as internal nodes in the tree, the number of terminal nodes and the training error rate.  
  First, I received an error as follows:  
  Text

  Description automatically generated  
  So, I did a little search and I came across a similar issue on the internet in [this](https://www.edureka.co/community/57829/random-forest-error-ymean-numeric-argument-binary-operator) website. The user suggested converting the variable from class to factor, so I tried that:  
  Text

  Description automatically generated
* Plotting a classification tree to display our tree structure and adding the node labels using the text() functions:  
  “The argument pretty=0 instructs R to include the category names for any qualitative predictors, rather than simply displaying a letter for each category.” *(ISLR, 325)*  
  Diagram

  Description automatically generated
* tree.carseats:  
  A screenshot of a computer

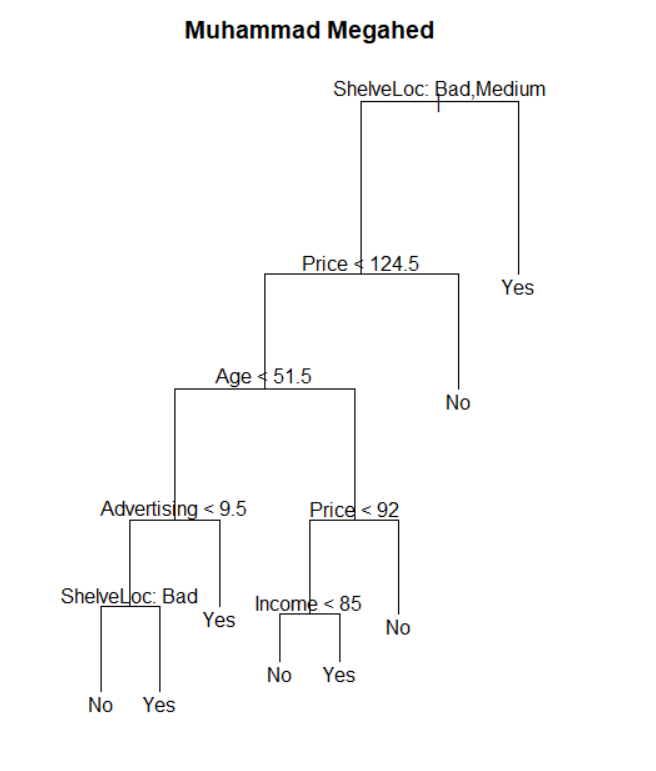
  Description automatically generated with medium confidence
* Properly evaluating the performance of the classification tree on the data by estimating the test error using the predict() function:  
  1. Splitting the observation into a training set and a test set.  
  2. Building the tree with the training set.  
  3. Evaluate its performance on the test set.  
  Text

  Description automatically generated with medium confidence
* Pruning the tree using class validation in hope to get better prediction results:  
  This process was done using the function cv.tree()  
  Graphical user interface, text

  Description automatically generated  
  “The argument “FUN = prune.misclass” indicates that we want the classification error rate to guide the cross-validation and pruning process instead of the default for the cv.tree() function which is deviance” *(ISLR, 326)*  
  $dev corresponds to the cross-validation error rate, and $size corresponds to the number of terminal nodes on the tree. So, we can say that the trees with 8 and 11 terminal nodes have the lowest cross-validation errors which is 48.
* Plotting the error rate as a function of both (size) and (k):  
  1. Plotting size.  
  2. Plotting k.  
  A picture containing calendar

  Description automatically generated  
  Chart, line chart

  Description automatically generated  
  3. Applying prune.misclass() to prune the tree and obtain the eight-node tree.
* Text

  Description automatically generated with medium confidence  
    
  4. Apply the predict function.  
  A picture containing text

  Description automatically generated

1. Fitting Regression Trees:

* Fitting a regression tree to the Boston data set using :  
  Text, letter

  Description automatically generated  
  Summary() function shows that 4 variables were used in construction the tree (rm, 1stat, crim, age)
* Plotting the tree:  
  A picture containing polygon

  Description automatically generated  
  Diagram, schematic

  Description automatically generated
* Using cv.tree() function to check the effect of the tree on performance if pursued:  
    
  Chart, line chart

  Description automatically generated  
  In this case the most complicated tree is selected for cross-validation. We can still prune the tree using the prune.tree() function.  
  Diagram

  Description automatically generated with low confidence  
  Diagram

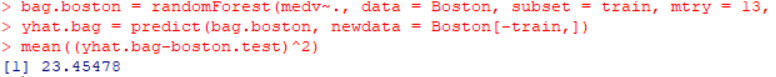
  Description automatically generated
* We use the unpruned tree to make predictions about the test set:  
  A picture containing text

  Description automatically generated  
  Chart, scatter chart

  Description automatically generated  
  This indicates that the test set MSE (Mean Squared Error) is 35.38. The square root of that is 5.948 which means that this model leads to test predictions that are within around $5,948 of the true maiden home value for the suburb.
* Bagging and Random Forests:  
  We apply bagging and random forests to the Boston data:  
  Graphical user interface, text

  Description automatically generated with medium confidence  
  mtry = 13 means that all 13 predictors should be considered every split of a tree (a.k.a bagging)
* Now testing how well it does:  
  A picture containing diagram

  Description automatically generated  
  Chart, scatter chart

  Description automatically generated
* Changing the number of trees grown by using “ntree” argument:  
    
  
* Using a smaller value of mtry:  
  A picture containing diagram

  Description automatically generated
* Using the importance() function:  
  Text, table

  Description automatically generated
* Using varImpPlot() to plot the importance measures:  
    
  Chart

  Description automatically generated

1. Boosting

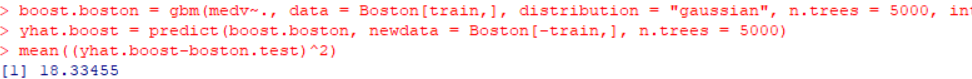
* Using the gbm package for boosting:  
  Graphical user interface, text

  Description automatically generated
* Chart, histogram

  Description automatically generatedProducing “partial independence plots” for the variables of 1stat and rm:  
  A picture containing text, device

  Description automatically generated  
  Chart, line chart

  Description automatically generated
* Using the boosted model to predict medv on the test:  
  
* Using a different value in the shrinkage parameter (default is 0.001, value used here is 0.2):



This leads to a slightly lower MSE value.