Prediction of House Prices in Boston

Aravind

April 1, 2018

Load libraries

```
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 3.4.4
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.4.4
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 3.4.4
```

```
## corrplot 0.84 loaded
```

```
library(Cubist)
```

```
## Warning: package 'Cubist' was built under R version 3.4.4
```

Data set Description

UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Housing (https://archive.ics.uci.edu/ml/datasets/Housing).

Each record in the database describes a Boston suburb or town. The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970. The attributes are de???ned as follows (taken from the UCI Machine Learning Repository): 1. CRIM: per capita crime rate by town 2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft. 3. INDUS: proportion of non-retail business acres per town 4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise) 5. NOX: nitric oxides concentration (parts per 10 million) 6. RM: average number of rooms per dwelling 7. AGE: proportion of owner-occupied units built prior to 1940 8. DIS: weighted distances to ???ve Boston employment centers 9. RAD: index of accessibility to radial highways 10. TAX: full-value property-tax rate per $10,000 11. PTRATIO: pupil-teacher ratio by town 12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town 13. LSTAT: % lower status of the population 14. MEDV: Median value of owner-occupied homes in $1000s

Problem Statement: To Predict the median house price in 1000 for suburbs in Boston.

1 Load the Dataset

The dataset is available in the mlbench package.

Attach the BostonHousing dataset

```
data(BostonHousing)
```

Split out validation dataset Create a list of 80% of the rows in the original dataset we can use for training and the reset for 20% for test

```
set.seed(7)
validation_index <- createDataPartition(BostonHousing$medv, p=0.80, list=FALSE)
validation <- BostonHousing[-validation_index,]
dataset <- BostonHousing[validation_index,]
```

2. Analyze Data

The objective of this step in the process is to better understand the problem.

2.1 Descriptive Statistics

```
# dimensions of dataset
dim(dataset)
```

```
## [1] 407  14
```

We have 407 instances to work with and can con???rm the data has 14 attributes including the class attribute medv.

Let’s also look at the data types of each attribute.

```
# list types for each attribute
sapply(dataset, class)
```

```
##      crim        zn     indus      chas       nox        rm       age 
## "numeric" "numeric" "numeric"  "factor" "numeric" "numeric" "numeric" 
##       dis       rad       tax   ptratio         b     lstat      medv 
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
```

We can see that one of the attributes (chas) is a factor while all of the others are numeric.

Let’s now take a peak at the ???rst 20 rows of the data.

```
# take a peek at the first 5 rows of the data
head(dataset, n=20)
```

```
##       crim   zn indus chas   nox    rm   age    dis rad tax ptratio      b
## 2  0.02731  0.0  7.07    0 0.469 6.421  78.9 4.9671   2 242    17.8 396.90
## 3  0.02729  0.0  7.07    0 0.469 7.185  61.1 4.9671   2 242    17.8 392.83
## 4  0.03237  0.0  2.18    0 0.458 6.998  45.8 6.0622   3 222    18.7 394.63
## 5  0.06905  0.0  2.18    0 0.458 7.147  54.2 6.0622   3 222    18.7 396.90
## 6  0.02985  0.0  2.18    0 0.458 6.430  58.7 6.0622   3 222    18.7 394.12
## 7  0.08829 12.5  7.87    0 0.524 6.012  66.6 5.5605   5 311    15.2 395.60
## 8  0.14455 12.5  7.87    0 0.524 6.172  96.1 5.9505   5 311    15.2 396.90
## 9  0.21124 12.5  7.87    0 0.524 5.631 100.0 6.0821   5 311    15.2 386.63
## 13 0.09378 12.5  7.87    0 0.524 5.889  39.0 5.4509   5 311    15.2 390.50
## 14 0.62976  0.0  8.14    0 0.538 5.949  61.8 4.7075   4 307    21.0 396.90
## 15 0.63796  0.0  8.14    0 0.538 6.096  84.5 4.4619   4 307    21.0 380.02
## 16 0.62739  0.0  8.14    0 0.538 5.834  56.5 4.4986   4 307    21.0 395.62
## 17 1.05393  0.0  8.14    0 0.538 5.935  29.3 4.4986   4 307    21.0 386.85
## 18 0.78420  0.0  8.14    0 0.538 5.990  81.7 4.2579   4 307    21.0 386.75
## 19 0.80271  0.0  8.14    0 0.538 5.456  36.6 3.7965   4 307    21.0 288.99
## 20 0.72580  0.0  8.14    0 0.538 5.727  69.5 3.7965   4 307    21.0 390.95
## 23 1.23247  0.0  8.14    0 0.538 6.142  91.7 3.9769   4 307    21.0 396.90
## 25 0.75026  0.0  8.14    0 0.538 5.924  94.1 4.3996   4 307    21.0 394.33
## 26 0.84054  0.0  8.14    0 0.538 5.599  85.7 4.4546   4 307    21.0 303.42
## 27 0.67191  0.0  8.14    0 0.538 5.813  90.3 4.6820   4 307    21.0 376.88
##    lstat medv
## 2   9.14 21.6
## 3   4.03 34.7
## 4   2.94 33.4
## 5   5.33 36.2
## 6   5.21 28.7
## 7  12.43 22.9
## 8  19.15 27.1
## 9  29.93 16.5
## 13 15.71 21.7
## 14  8.26 20.4
## 15 10.26 18.2
## 16  8.47 19.9
## 17  6.58 23.1
## 18 14.67 17.5
## 19 11.69 20.2
## 20 11.28 18.2
## 23 18.72 15.2
## 25 16.30 15.6
## 26 16.51 13.9
## 27 14.81 16.6
```

Let’s summarize the distribution of each attribute.

```
# summarize attribute distributions
summary(dataset)
```

```
##       crim                zn             indus       chas
##  Min.   : 0.00906   Min.   : 0.00   Min.   : 0.46   0:376
##  1st Qu.: 0.08556   1st Qu.: 0.00   1st Qu.: 5.19   1: 31
##  Median : 0.28955   Median : 0.00   Median : 9.90
##  Mean   : 3.58281   Mean   :10.57   Mean   :11.36
##  3rd Qu.: 3.50464   3rd Qu.: 0.00   3rd Qu.:18.10
##  Max.   :88.97620   Max.   :95.00   Max.   :27.74
##       nox               rm             age              dis
##  Min.   :0.3850   Min.   :3.863   Min.   :  2.90   Min.   : 1.130
##  1st Qu.:0.4530   1st Qu.:5.873   1st Qu.: 45.05   1st Qu.: 2.031
##  Median :0.5380   Median :6.185   Median : 77.70   Median : 3.216
##  Mean   :0.5577   Mean   :6.279   Mean   : 68.83   Mean   : 3.731
##  3rd Qu.:0.6310   3rd Qu.:6.611   3rd Qu.: 94.55   3rd Qu.: 5.100
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :10.710
##       rad              tax           ptratio            b
##  Min.   : 1.000   Min.   :188.0   Min.   :12.60   Min.   :  0.32
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:374.50
##  Median : 5.000   Median :330.0   Median :19.00   Median :391.13
##  Mean   : 9.464   Mean   :405.6   Mean   :18.49   Mean   :357.88
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.27
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90
##      lstat             medv
##  Min.   : 1.730   Min.   : 5.00
##  1st Qu.: 6.895   1st Qu.:17.05
##  Median :11.500   Median :21.20
##  Mean   :12.827   Mean   :22.61
##  3rd Qu.:17.175   3rd Qu.:25.00
##  Max.   :37.970   Max.   :50.00
```

We can note that chas is a pretty unbalanced factor. We could transform this attribute to numeric to make calculating descriptive statistics and plots easier.

```
# convert factor to numeric
dataset[,4] <- as.numeric(as.character(dataset[,4]))
```

Now, let’s now take a look at the correlation between all of the numeric attributes.

```
# summarize correlations between input variables
cor(dataset[,1:13])
```

```
##                crim          zn       indus        chas        nox
## crim     1.00000000 -0.19790631  0.40597009 -0.05713065  0.4232413
## zn      -0.19790631  1.00000000 -0.51895069 -0.04843477 -0.5058512
## indus    0.40597009 -0.51895069  1.00000000  0.08003629  0.7665481
## chas    -0.05713065 -0.04843477  0.08003629  1.00000000  0.1027366
## nox      0.42324132 -0.50585121  0.76654811  0.10273656  1.0000000
## rm      -0.21513269  0.28942883 -0.37673408  0.08252441 -0.2988506
## age      0.35438190 -0.57070265  0.65858310  0.10938121  0.7238371
## dis     -0.39050970  0.65618742 -0.72305885 -0.11142420 -0.7708680
## rad      0.64240501 -0.29952976  0.56774365 -0.00901245  0.5851676
## tax      0.60622608 -0.28791668  0.68070916 -0.02779018  0.6521787
## ptratio  0.28929828 -0.35341215  0.32920610 -0.13554380  0.1416616
## b       -0.30211854  0.16927489 -0.33597951  0.04724420 -0.3620791
## lstat    0.47537617 -0.39712686  0.59212718 -0.04569239  0.5819645
##                  rm        age        dis         rad         tax
## crim    -0.21513269  0.3543819 -0.3905097  0.64240501  0.60622608
## zn       0.28942883 -0.5707027  0.6561874 -0.29952976 -0.28791668
## indus   -0.37673408  0.6585831 -0.7230588  0.56774365  0.68070916
## chas     0.08252441  0.1093812 -0.1114242 -0.00901245 -0.02779018
## nox     -0.29885055  0.7238371 -0.7708680  0.58516760  0.65217875
## rm       1.00000000 -0.2325359  0.1952159 -0.19149122 -0.26794733
## age     -0.23253586  1.0000000 -0.7503321  0.45235421  0.50164657
## dis      0.19521590 -0.7503321  1.0000000 -0.49382744 -0.52649325
## rad     -0.19149122  0.4523542 -0.4938274  1.00000000  0.92137876
## tax     -0.26794733  0.5016466 -0.5264932  0.92137876  1.00000000
## ptratio -0.32000372  0.2564318 -0.2021897  0.45312318  0.44192428
## b        0.15539923 -0.2512574  0.2826819 -0.41033069 -0.41848779
## lstat   -0.62038075  0.5932128 -0.4957302  0.47306604  0.52339243
##            ptratio          b       lstat
## crim     0.2892983 -0.3021185  0.47537617
## zn      -0.3534121  0.1692749 -0.39712686
## indus    0.3292061 -0.3359795  0.59212718
## chas    -0.1355438  0.0472442 -0.04569239
## nox      0.1416616 -0.3620791  0.58196447
## rm      -0.3200037  0.1553992 -0.62038075
## age      0.2564318 -0.2512574  0.59321281
## dis     -0.2021897  0.2826819 -0.49573024
## rad      0.4531232 -0.4103307  0.47306604
## tax      0.4419243 -0.4184878  0.52339243
## ptratio  1.0000000 -0.1495283  0.35375936
## b       -0.1495283  1.0000000 -0.37661571
## lstat    0.3537594 -0.3766157  1.00000000
```

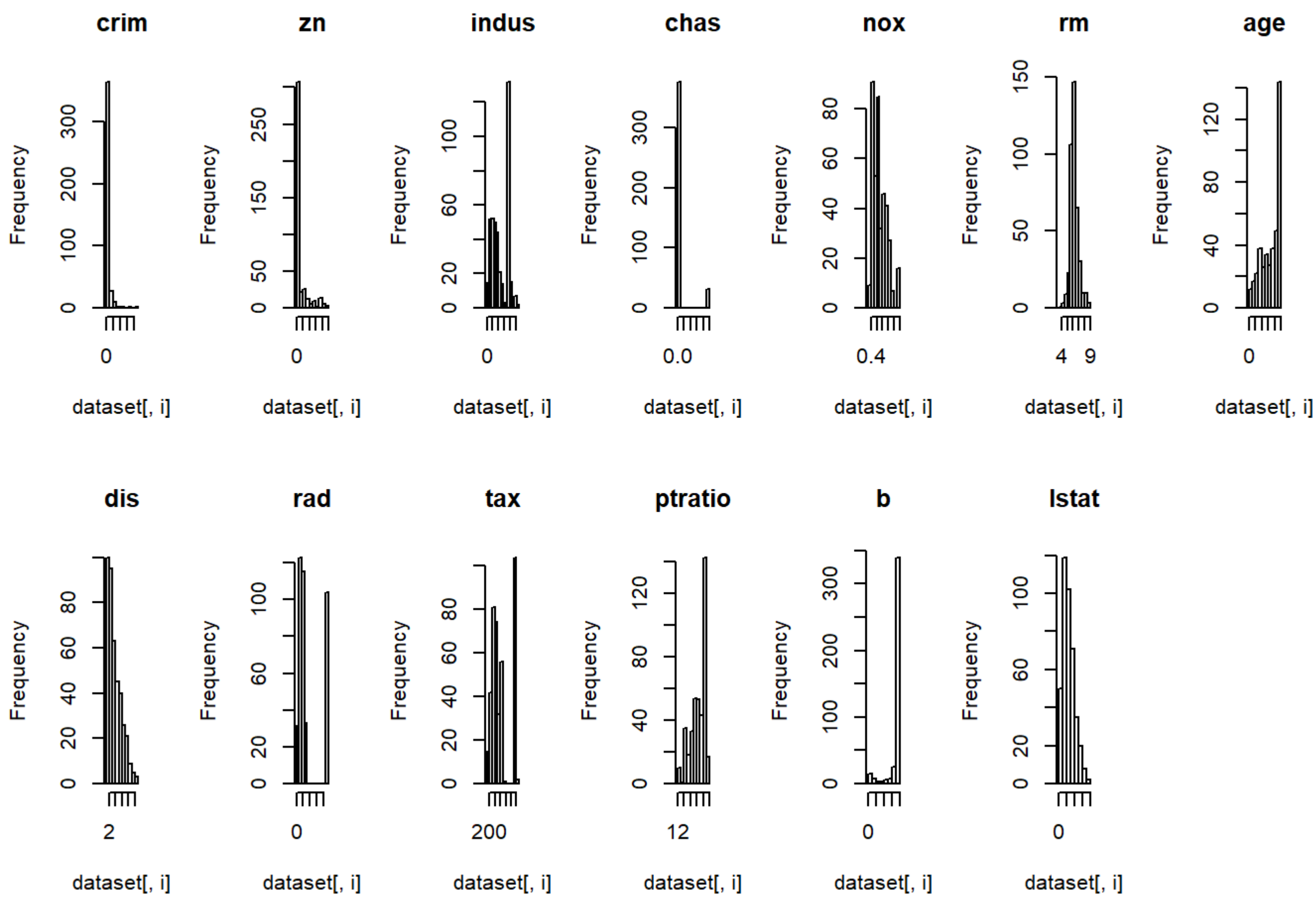
We can see that many of the attributes have a strong correlation (e.g. > 0.70 or < 0.70). For example: nox and indus with 0.77. dist and indus with 0.71. tax and indus with 0.72. age and nox with 0.72. dist and nox with 0.76.

This is collinearity and we may see better results with regression algorithms if the correlated attributes are removed.

2.2 Unimodal Data Visualizations

Let's look at histograms of each attribute to get a sense of the data distributions.

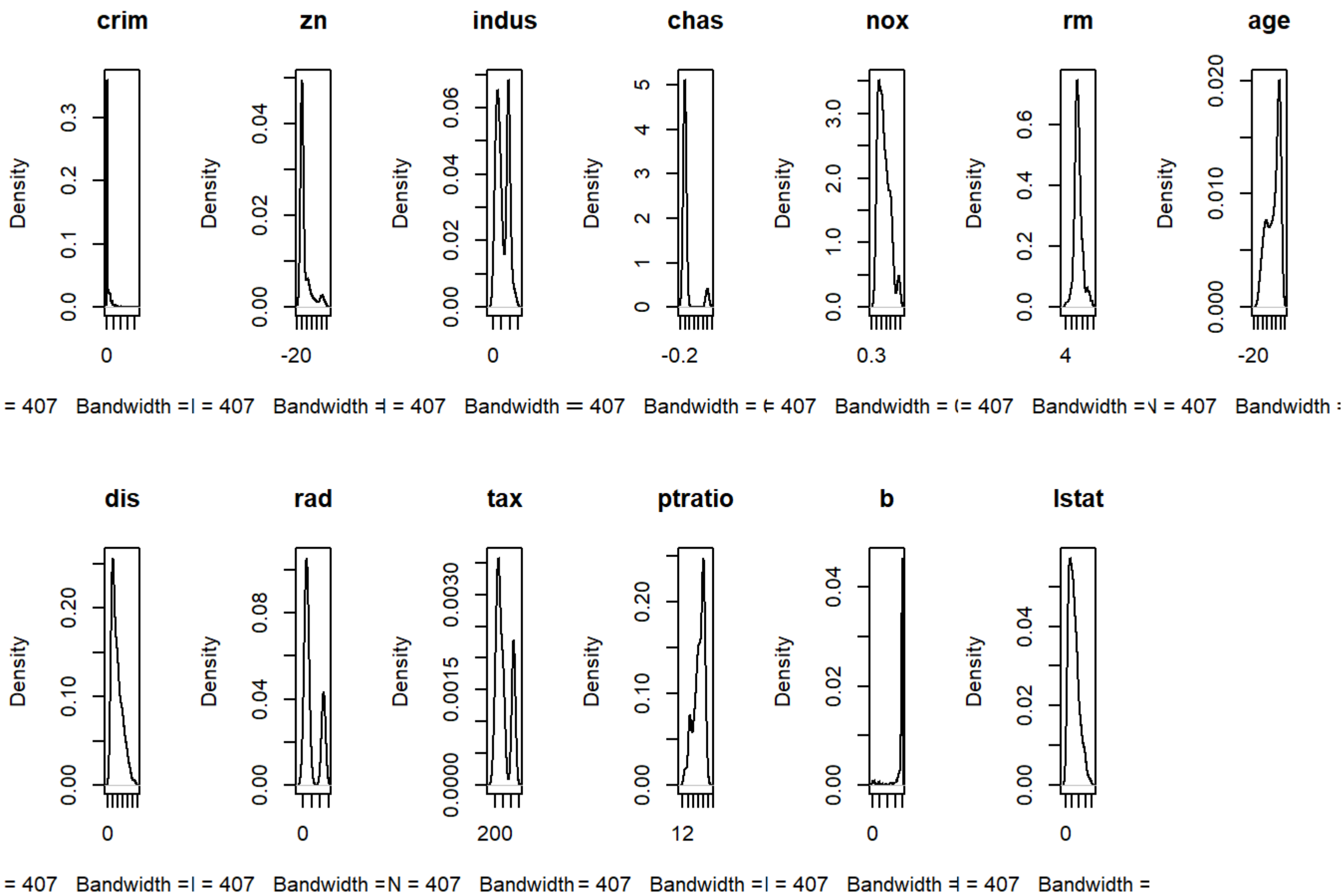
```r
# histograms each attribute
par(mfrow=c(2,7))
for(i in 1:13) {
    hist(dataset[,i], main=names(dataset)[i])
}
```



We can see that some attributes may have an exponential distribution, such as crim, zn, ange and b. We can see that others may have a bimodal distribution such as rad and tax.

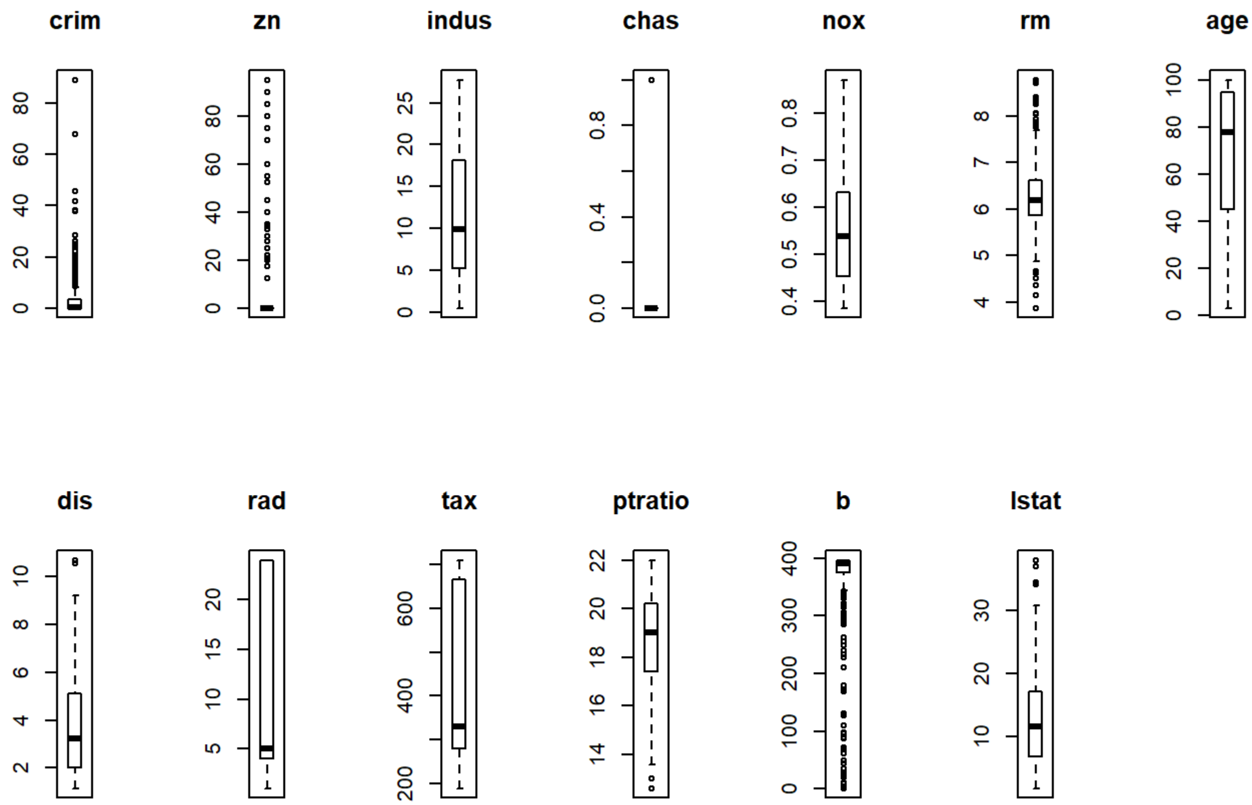
Let’s look at the same distributions using density plots that smooth them out a bit.

```r
# density plot for each attribute
par(mfrow=c(2,7))
for(i in 1:13) {
    plot(density(dataset[,i]), main=names(dataset)[i])
}
```



Let’s look at the data with box and whisker plots of each attribute

```
# boxplots for each attribute
par(mfrow=c(2,7))
for(i in 1:13) {
    boxplot(dataset[,i], main=names(dataset)[i])
}
```



This helps point out the skew in many distributions so much so that data looks like outliers

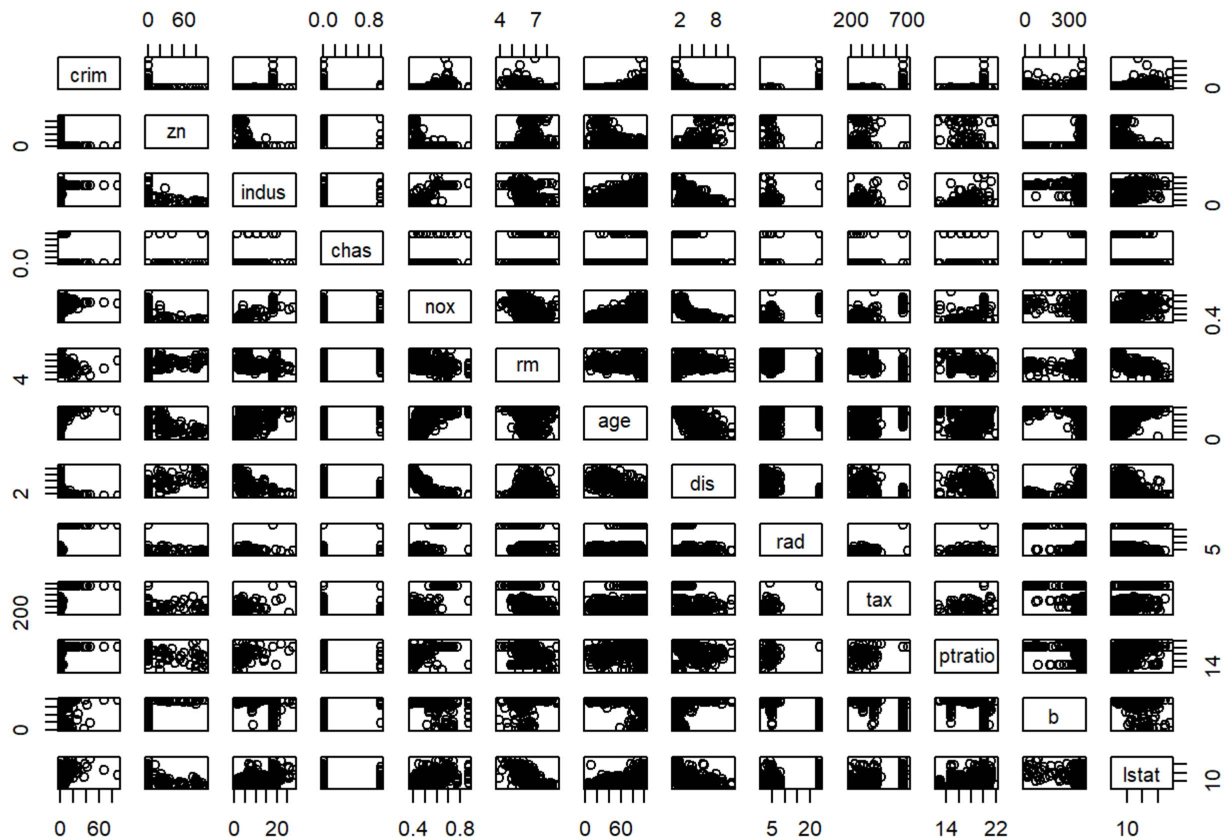
The larger darker blue dots con???rm the positively correlated attributes we listed early (not the diagonal). We can also see some larger darker red dots that suggest some negatively correlated attributes. For example tax and rad. These too may be candidates for removal to better improve accuracy of models later on.

2.3 Multi modal Data Visualizations

Let's look at some visualizations of the interactions between variables.

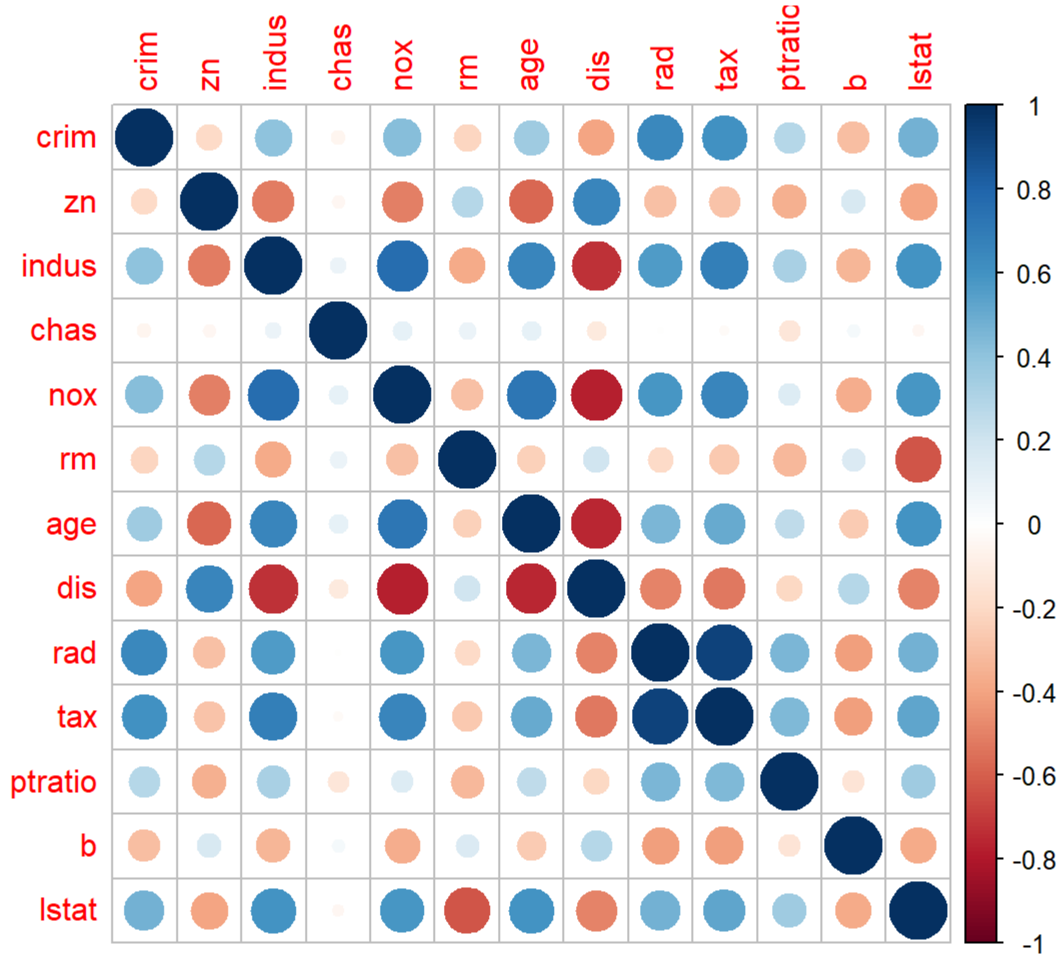
The best place to start is a scatterplot matrix.

```
# scatterplot matrix
pairs(dataset[,1:13])
```



We can see that some of the higher correlated attributes do show good structure in their relationship. Not linear, but nice predictable curved relationships.

```
# correlation plot
correlations <- cor(dataset[,1:13])
corrplot(correlations, method="circle")
```



2.4 Summary of Ideas

There is a lot of structure in this dataset. We need to think about transforms that we could use later to better expose the structure which in turn may improve modeling accuracy. So far it would be worth trying: Feature selection and removing the most correlated attributes. Normalizing the dataset to reduce the e???ect of di???ering scales. Standardizing the dataset to reduce the e???ects of di???ering distributions. Box-Cox transform to see if ???attening out some of the distributions improves accuracy.

3 Evaluate Algorithms: Baseline

We have no idea what algorithms will do well on this problem. Gut feel suggests regression algorithms like GLM and GLMNET may do well. It is also possible that decision trees and even SVM may do well.

We will use 10-fold cross validation (each fold will be about 360 instances for training and 40 for test) with 3 repeats. The dataset is not too small and this is a good standard test harness con???guration. We will evaluate algorithms using the RMSE and R2 metrics. RMSE will give a gross idea of how wrong all predictions are (0 is perfect) and R2 will give an idea of how well the model has ???t the data (1 is perfect, 0 is worst).

```
# Run algorithms using 10-fold cross validation
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
```

Let's create a baseline of performance on this problem and spot-check a number of di???erent algorithms. We will select a suite of di???erent algorithms capable of working on this regression problem. The 6 algorithms selected include: Linear Algorithms: Linear Regression (LR), Generalized Linear Regression (GLM) and Penalized Linear Regression (GLMNET) Non-Linear Algorithms: Classi???cation and Regression Trees (CART), Support Vector

Machines (SVM) with a radial basis function and k-Nearest Neighbors (KNN) We know the data has di???ering units of measure so we will standardize the data for this baseline comparison. This will those algorithms that prefer data in the same scale (e.g. instance based methods and some regression algorithms) a chance to do well.

```
# lm
set.seed(7)
fit.lm <- train(medv~., data=dataset, method="lm", metric=metric, preProc=c("center", "scale"),
 trControl=control)
# GLM
set.seed(7)
fit.glm <- train(medv~., data=dataset, method="glm", metric=metric, preProc=c("center", "scale"
), trControl=control)
# GLMNET
set.seed(7)
fit.glmnet <- train(medv~., data=dataset, method="glmnet", metric=metric, preProc=c("center", "s
cale"), trControl=control)
# SVM
set.seed(7)
fit.svm <- train(medv~., data=dataset, method="svmRadial", metric=metric, preProc=c("center", "s
cale"), trControl=control)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.cart <- train(medv~., data=dataset, method="rpart", metric=metric, tuneGrid=grid, preProc=c(
"center", "scale"), trControl=control)
# kNN
set.seed(7)
fit.knn <- train(medv~., data=dataset, method="knn", metric=metric, preProc=c("center", "scale"
), trControl=control)
```

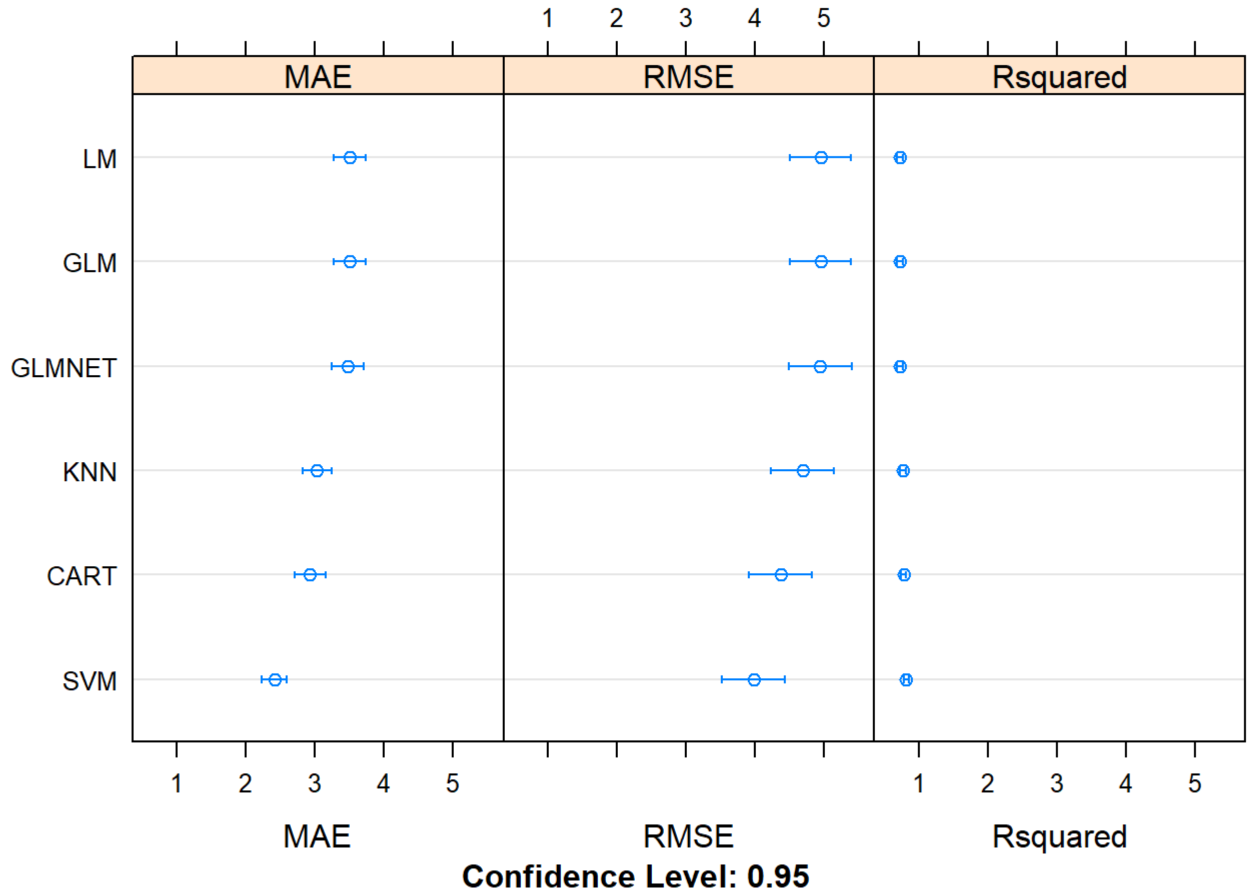
The algorithms all use default tuning parameters, except CART which is fussy on this dataset and has 3 default parameters speci???ed.

Let’s compare the algorithms.

```
# Compare algorithms
results <- resamples(list(LM=fit.lm, GLM=fit.glm, GLMNET=fit.glmnet, SVM=fit.svm, CART=fit.cart,
 KNN=fit.knn))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: LM, GLM, GLMNET, SVM, CART, KNN
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.275798 3.105566 3.476184 3.501075 3.988069 4.806931    0
## GLM    2.275798 3.105566 3.476184 3.501075 3.988069 4.806931    0
## GLMNET 2.249747 3.054908 3.436463 3.472943 4.002405 4.819706    0
## SVM    1.636837 2.083861 2.395264 2.412779 2.615126 3.597281    0
## CART   2.043357 2.509889 2.875049 2.931782 3.294371 4.707060    0
## KNN    2.036111 2.675617 3.023173 3.026927 3.346350 4.663144    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.945061 4.120871 4.590366 4.946754 5.622836 7.966565    0
## GLM    2.945061 4.120871 4.590366 4.946754 5.622836 7.966565    0
## GLMNET 2.900783 4.135902 4.580667 4.944112 5.641409 8.044256    0
## SVM    2.164717 2.979377 3.736229 3.980410 4.723911 7.251589    0
## CART   2.519930 3.428198 4.294325 4.372673 5.169167 7.856569    0
## KNN    3.100655 3.690470 4.579753 4.687796 5.442480 8.369137    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LM     0.4723270 0.6698297 0.7479362 0.7323877 0.8184113 0.8830690    0
## GLM    0.4723270 0.6698297 0.7479362 0.7323877 0.8184113 0.8830690    0
## GLMNET 0.4602065 0.6704187 0.7505348 0.7325322 0.8182185 0.8904240    0
## SVM    0.5715369 0.7795105 0.8572078 0.8249574 0.8968088 0.9474025    0
## CART   0.4803188 0.7100007 0.7991753 0.7837238 0.8504914 0.9358565    0
## KNN    0.4295926 0.7514504 0.7924037 0.7746890 0.8521054 0.9166735    0
```

```
dotplot(results)
```



It looks like SVM has the lowest RMSE, followed closely by the other non-linear algorithms CART and KNN. The linear regression algorithms all appear to be in the same ball park and slightly worse error.

We can also see that SVM and the other non-linear algorithms have the best ???t for the data in their R2 measures

4 Evaluate Algorithms: Feature Selection

We have a theory that the correlated attributes are reducing the accuracy of the linear algorithms tried in the base line spot-check in the last step. In this step we will remove the highly correlated attributes and see what e???ect that has on the evaluation metrics. We can ???nd and remove the highly correlated attributes using the findCorrelation() function from the caret package

```
# remove correlated attributes
# find attributes that are highly corrected
set.seed(7)
cutoff <- 0.70
correlations <- cor(dataset[,1:13])
highlyCorrelated <- findCorrelation(correlations, cutoff=cutoff)
for (value in highlyCorrelated) {
  print(names(dataset)[value])
}
```

```
## [1] "indus"
## [1] "nox"
## [1] "tax"
## [1] "dis"
```

```
# create a new dataset without highly corrected features
dataset_features <- dataset[,-highlyCorrelated]
dim(dataset_features)
```

```
## [1] 407  10
```

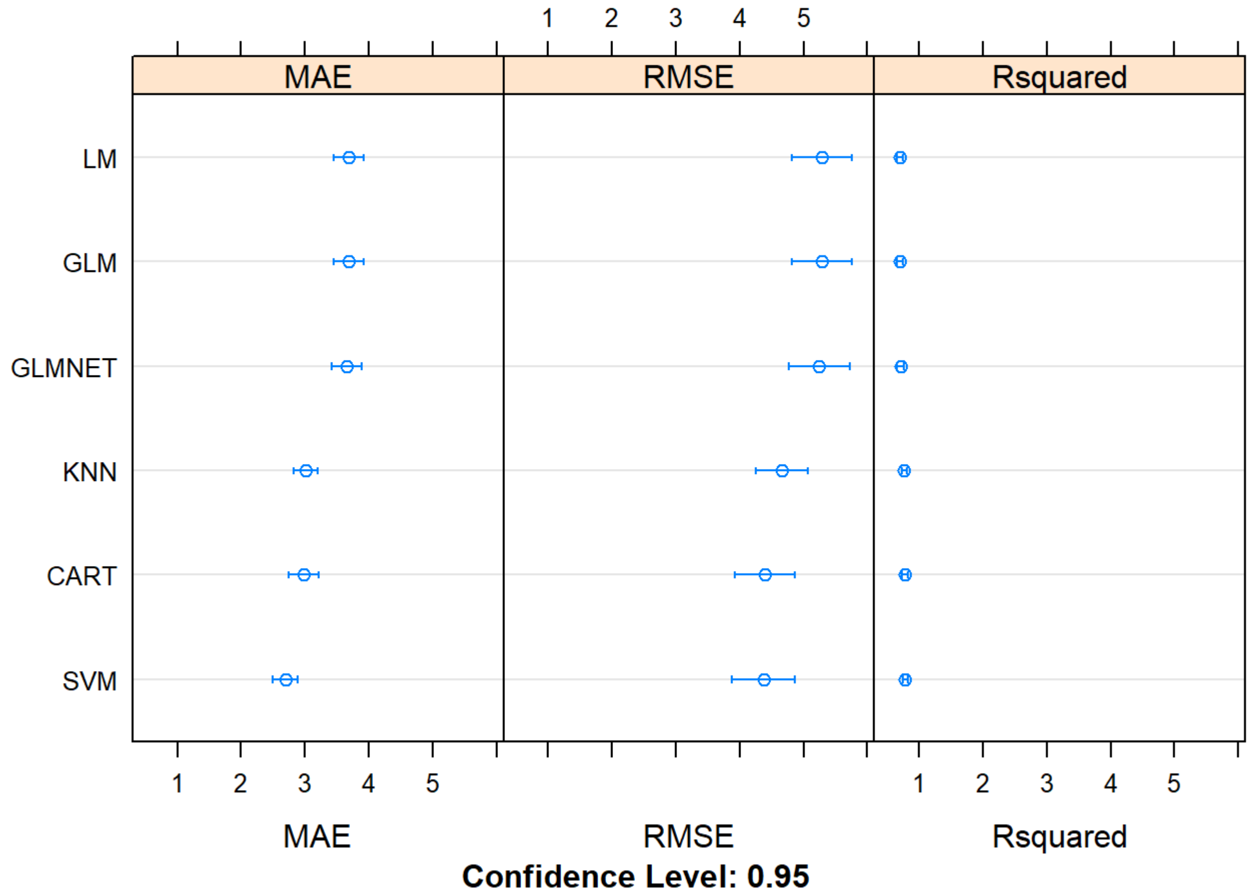
We can see that we have dropped 4 attributes: indus, box, tax and dis.

Now let’s try the same 6 algorithms from our base line experiment.

```
# Run algorithms using 10-fold cross validation
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
# lm
set.seed(7)
fit.lm <- train(medv~., data=dataset_features, method="lm", metric=metric, preProc=c("center",
"scale"), trControl=control)
# GLM
set.seed(7)
fit.glm <- train(medv~., data=dataset_features, method="glm", metric=metric, preProc=c("center",
 "scale"), trControl=control)
# GLMNET
set.seed(7)
fit.glmnet <- train(medv~., data=dataset_features, method="glmnet", metric=metric, preProc=c("ce
nter", "scale"), trControl=control)
# SVM
set.seed(7)
fit.svm <- train(medv~., data=dataset_features, method="svmRadial", metric=metric, preProc=c("ce
nter", "scale"), trControl=control)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.cart <- train(medv~., data=dataset_features, method="rpart", metric=metric, tuneGrid=grid, p
reProc=c("center", "scale"), trControl=control)
# kNN
set.seed(7)
fit.knn <- train(medv~., data=dataset_features, method="knn", metric=metric, preProc=c("center",
 "scale"), trControl=control)
# Compare algorithms
feature_results <- resamples(list(LM=fit.lm, GLM=fit.glm, GLMNET=fit.glmnet, SVM=fit.svm, CART=f
it.cart, KNN=fit.knn))
summary(feature_results)
```

```
##
## Call:
## summary.resamples(object = feature_results)
##
## Models: LM, GLM, GLMNET, SVM, CART, KNN
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.581207 3.245913 3.612843 3.687229 4.026392 5.148788    0
## GLM    2.581207 3.245913 3.612843 3.687229 4.026392 5.148788    0
## GLMNET 2.457098 3.263388 3.612401 3.655058 3.866505 5.154045    0
## SVM    1.701193 2.352631 2.756603 2.692734 3.070049 3.998602    0
## CART   2.244475 2.512546 2.867973 2.977266 3.153672 5.091982    0
## KNN    2.175500 2.661646 3.047287 3.014089 3.331220 4.500488    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     3.240150 4.366222 4.991584 5.285267 5.903958 8.517783    0
## GLM    3.240150 4.366222 4.991584 5.285267 5.903958 8.517783    0
## GLMNET 2.958713 4.436144 4.983526 5.247147 5.986477 8.480235    0
## SVM    2.283553 3.333524 4.041464 4.375623 5.496401 7.577551    0
## CART   2.897772 3.492845 4.118793 4.395251 4.617383 8.171875    0
## KNN    3.144079 3.908360 4.546349 4.662776 5.571105 7.883717    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LM     0.4129148 0.6386145 0.7038130 0.6979964 0.7878433 0.8720645    0
## GLM    0.4129148 0.6386145 0.7038130 0.6979964 0.7878433 0.8720645    0
## GLMNET 0.3986900 0.6329603 0.7241796 0.7057707 0.8139213 0.8833758    0
## SVM    0.5272147 0.6978457 0.8297009 0.7824937 0.8750123 0.9259037    0
## CART   0.4651813 0.7218025 0.8167862 0.7782884 0.8608693 0.9123469    0
## KNN    0.4870866 0.6954455 0.7682838 0.7630308 0.8542543 0.8962957    0
```

```
dotplot(feature_results)
```



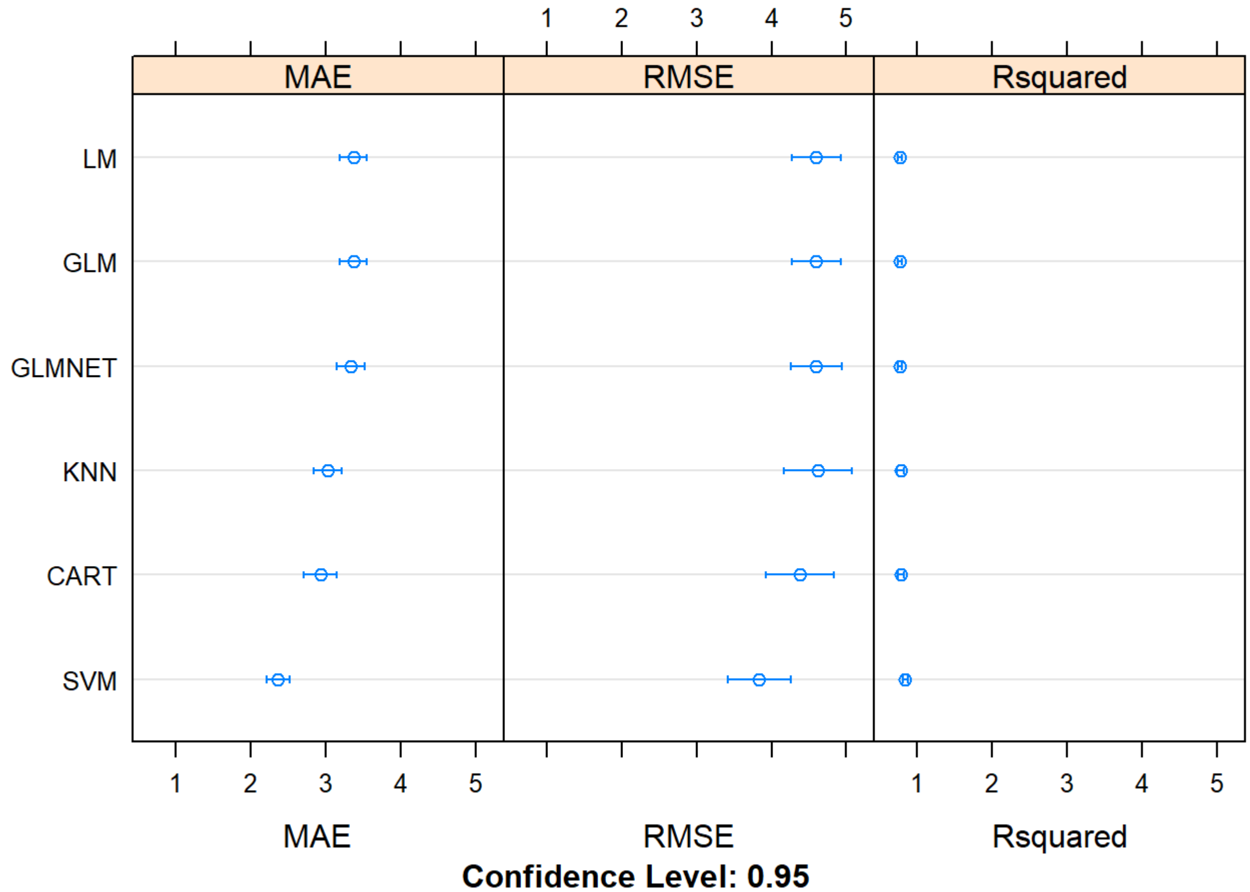
Comparing the results, we can see that this has made the RMSE worse for the linear and the non-linear algorithms. The correlated attributes we removed are contributing to the accuracy of the models.

5 Evaluate Algorithms: Box-Cox Transform

```
# Run algorithms using 10-fold cross validation
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
# lm
set.seed(7)
fit.lm <- train(medv~., data=dataset, method="lm", metric=metric, preProc=c("center", "scale",
"BoxCox"), trControl=control)
# GLM
set.seed(7)
fit.glm <- train(medv~., data=dataset, method="glm", metric=metric, preProc=c("center", "scale",
 "BoxCox"), trControl=control)
# GLMNET
set.seed(7)
fit.glmnet <- train(medv~., data=dataset, method="glmnet", metric=metric, preProc=c("center", "s
cale", "BoxCox"), trControl=control)
# SVM
set.seed(7)
fit.svm <- train(medv~., data=dataset, method="svmRadial", metric=metric, preProc=c("center", "s
cale", "BoxCox"), trControl=control)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.cart <- train(medv~., data=dataset, method="rpart", metric=metric, tuneGrid=grid, preProc=c(
"center", "scale", "BoxCox"), trControl=control)
# kNN
set.seed(7)
fit.knn <- train(medv~., data=dataset, method="knn", metric=metric, preProc=c("center", "scale",
 "BoxCox"), trControl=control)
# Compare algorithms
transform_results <- resamples(list(LM=fit.lm, GLM=fit.glm, GLMNET=fit.glmnet, SVM=fit.svm, CART
=fit.cart, KNN=fit.knn))
summary(transform_results)
```

```
##
## Call:
## summary.resamples(object = transform_results)
##
## Models: LM, GLM, GLMNET, SVM, CART, KNN
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.520337 2.963410 3.385355 3.371279 3.611259 4.311583    0
## GLM    2.520337 2.963410 3.385355 3.371279 3.611259 4.311583    0
## GLMNET 2.505158 2.922326 3.328423 3.334650 3.596593 4.377119    0
## SVM    1.616495 2.143104 2.342498 2.368048 2.642978 3.443715    0
## CART   2.043357 2.509889 2.875049 2.934449 3.294371 4.707060    0
## KNN    2.193333 2.641599 3.140515 3.034677 3.300678 4.373442    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     3.327180 3.936074 4.441338 4.600223 5.183245 6.733523    0
## GLM    3.327180 3.936074 4.441338 4.600223 5.183245 6.733523    0
## GLMNET 3.373948 3.925110 4.431670 4.595578 5.186260 6.877090    0
## SVM    2.301160 2.881686 3.601308 3.836561 4.498855 7.014149    0
## CART   2.563901 3.428198 4.294325 4.374859 5.169167 7.856569    0
## KNN    3.172856 3.757953 4.464893 4.619555 5.116853 8.451237    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LM     0.5972973 0.7270836 0.7727291 0.7633673 0.8114439 0.8901553    0
## GLM    0.5972973 0.7270836 0.7727291 0.7633673 0.8114439 0.8901553    0
## GLMNET 0.5928990 0.7276325 0.7765796 0.7641157 0.8171126 0.8912449    0
## SVM    0.6000736 0.7997356 0.8603609 0.8374185 0.8892423 0.9533119    0
## CART   0.4803188 0.7100007 0.7991753 0.7834822 0.8504914 0.9333419    0
## KNN    0.4128347 0.7518341 0.8064478 0.7754822 0.8439104 0.9195463    0
```

```
dotplot(transform_results)
```



We can see that this indeed decrease the RMSE and increased the R2 on all except the CART algorithms. The RMSE of SVM dropped to an average of 3.761.

6 Improve Results With Tuning

We can improve the accuracy of the well performing algorithms by tuning their parameters. In this section we will look at tuning the parameters of SVM with a Radial Basis Function (RBF). with more time it might be worth exploring tuning of the parameters for CART and KNN. It might also be worth exploring other kernels for SVM besides the RBF. Let's look at the default parameters already adopted.

```
# look at parameters
print(fit.svm)
```

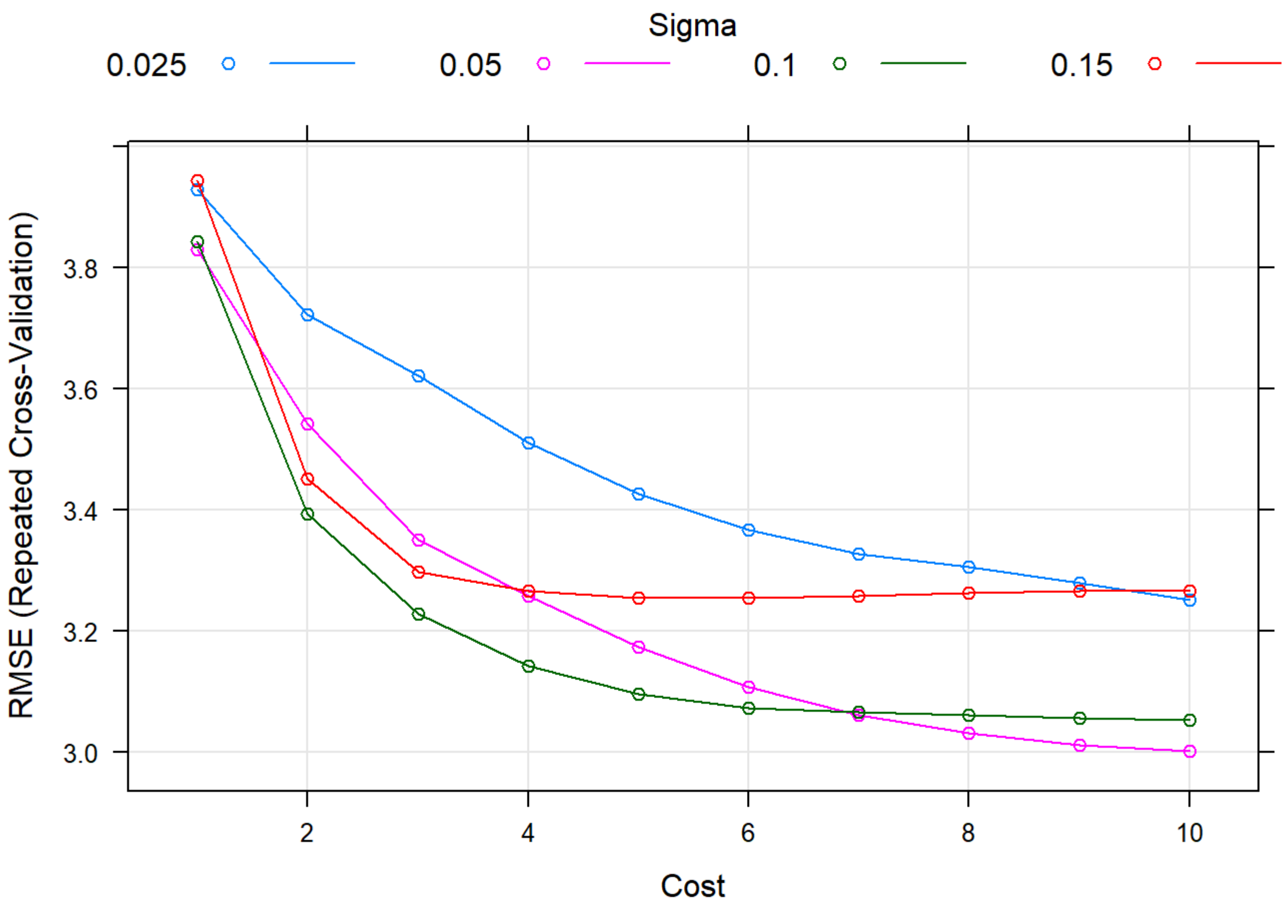
```
## Support Vector Machines with Radial Basis Function Kernel
##
## 407 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13), Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 366, 367, 366, 367, 365, 367, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##   0.25  4.692364  0.7795827  2.776964
##   0.50  4.251248  0.8083441  2.524725
##   1.00  3.836561  0.8374185  2.368048
##
## Tuning parameter 'sigma' was held constant at a value of 0.08878349
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.08878349 and C = 1.
```

```
# tune SVM sigma and C parametres
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
set.seed(7)
grid <- expand.grid(.sigma=c(0.025, 0.05, 0.1, 0.15), .C=seq(1, 10, by=1))
fit.svm <- train(medv~., data=dataset, method="svmRadial", metric=metric, tuneGrid=grid, preProc
=c("BoxCox"), trControl=control)
print(fit.svm)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 407 samples
##  13 predictor
##
## Pre-processing: Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 366, 367, 366, 367, 365, 367, ...
## Resampling results across tuning parameters:
##
##   sigma  C   RMSE      Rsquared   MAE
##   0.025   1  3.930458  0.8276275  2.439609
##   0.025   2  3.723009  0.8409733  2.328927
##   0.025   3  3.621614  0.8482064  2.280724
##   0.025   4  3.510280  0.8565961  2.231254
##   0.025   5  3.426241  0.8629571  2.203999
##   0.025   6  3.366889  0.8675244  2.184939
##   0.025   7  3.327847  0.8706149  2.173270
##   0.025   8  3.306757  0.8721949  2.165520
##   0.025   9  3.278829  0.8742127  2.155252
##   0.025  10  3.251596  0.8762207  2.144621
##   0.050   1  3.830832  0.8363320  2.356105
##   0.050   2  3.543118  0.8554896  2.254228
##   0.050   3  3.350911  0.8689186  2.173351
##   0.050   4  3.258575  0.8755807  2.134096
##   0.050   5  3.174041  0.8815445  2.092097
##   0.050   6  3.107810  0.8861029  2.063514
##   0.050   7  3.061503  0.8892309  2.049936
##   0.050   8  3.030597  0.8913244  2.042967
##   0.050   9  3.011698  0.8926385  2.039596
##   0.050  10  3.001111  0.8933696  2.039105
##   0.100   1  3.843944  0.8374210  2.369040
##   0.100   2  3.393117  0.8671391  2.176772
##   0.100   3  3.227843  0.8775505  2.104811
##   0.100   4  3.142971  0.8836300  2.081783
##   0.100   5  3.095937  0.8869367  2.066293
##   0.100   6  3.073112  0.8885969  2.056102
##   0.100   7  3.066507  0.8891503  2.054790
##   0.100   8  3.060573  0.8896855  2.054430
##   0.100   9  3.055764  0.8900990  2.056769
##   0.100  10  3.053300  0.8903036  2.061258
##   0.150   1  3.943847  0.8317027  2.387425
##   0.150   2  3.451454  0.8626185  2.204039
##   0.150   3  3.298200  0.8729186  2.143969
##   0.150   4  3.265943  0.8759410  2.141796
##   0.150   5  3.255373  0.8766262  2.142137
##   0.150   6  3.254155  0.8767197  2.151371
##   0.150   7  3.257504  0.8764573  2.164027
##   0.150   8  3.263358  0.8759398  2.176578
##   0.150   9  3.265779  0.8756556  2.188381
##   0.150  10  3.266036  0.8754983  2.197180
##
```

```
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.05 and C = 10.
```

```
plot(fit.svm)
```



The C parameter is the cost constraint used by SVM. Learn more in the help for the ksvm function ?ksvm. We can see from previous results that a C value of 1.0 is a good starting point.

Let's design a grid search around a C value of 1. We might see a small trend of decreasing RMSE with increasing C, so lets try all integer C values between 1 and 10. Another parameter that caret lets us tune is the sigma parameter. This is a smoothing parameter. Good sigma values are often start around 0.1, so we will try numbers before and after.

We can see that the sigma values ???atten out with larger C cost constraints. It looks like we might do well with a sigma of 0.05 and a C of 10. This gives us a respectable RMSE of 2.977085

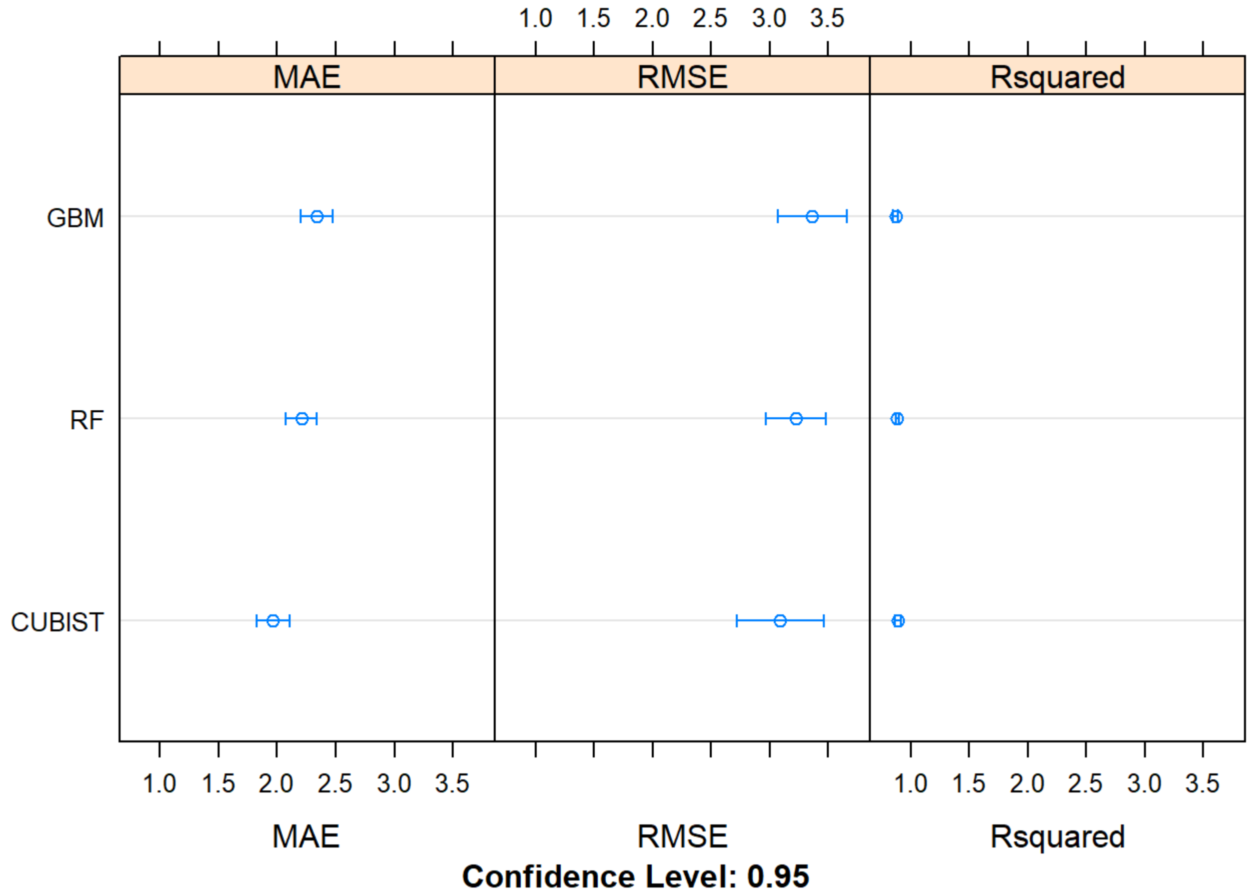
7 Ensemble Methods

19.7 Ensemble Methods We can try some ensemble methods on the problem and see if we can get a further decrease in our RMSE. In this section we will look at some boosting and bagging techniques for decision trees. Additional approaches you could look into would be blending the predictions of multiple well performing models together, called stacking. Let's take a look at the following ensemble methods: □ Random Forest, bagging (RF). □ Gradient Boosting Machines boosting (GBM). □ Cubist, boosting (CUBIST).

```
# try ensembles
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
# Random Forest
set.seed(7)
fit.rf <- train(medv~., data=dataset, method="rf", metric=metric, preProc=c("BoxCox"), trControl
=control)
# Stochastic Gradient Boosting
set.seed(7)
fit.gbm <- train(medv~., data=dataset, method="gbm", metric=metric, preProc=c("BoxCox"), trContr
ol=control, verbose=FALSE)
# Cubist
set.seed(7)
fit.cubist <- train(medv~., data=dataset, method="cubist", metric=metric, preProc=c("BoxCox"), t
rControl=control)
# Compare algorithms
ensemble_results <- resamples(list(RF=fit.rf, GBM=fit.gbm, CUBIST=fit.cubist))
summary(ensemble_results)
```

```
##
## Call:
## summary.resamples(object = ensemble_results)
##
## Models: RF, GBM, CUBIST
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## RF     1.562720 1.977781 2.177467 2.206746 2.355216 3.002479    0
## GBM    1.684321 2.055839 2.296176 2.337663 2.488155 3.073921    0
## CUBIST 1.225568 1.631722 1.998192 1.965454 2.248008 2.652877    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## RF     2.324293 2.779604 3.052795 3.220947 3.548645 5.091085    0
## GBM    2.296971 2.684094 3.146213 3.363994 4.213876 4.949682    0
## CUBIST 1.624740 2.323381 2.765466 3.089250 3.847823 5.259804    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF     0.8015000 0.8563858 0.8905490 0.8857147 0.9202120 0.9472200    0
## GBM    0.7395492 0.8346060 0.8798001 0.8702191 0.9127787 0.9603358    0
## CUBIST 0.6764145 0.8541497 0.9083048 0.8888146 0.9361016 0.9692201    0
```

```
dotplot(ensemble_results)
```



We can see that Cubist was the most accurate with an RMSE that was lower than that achieved by tuning SVM.

```
# Tune Cubist

# look at parameters used for Cubist
print(fit.cubist)
```

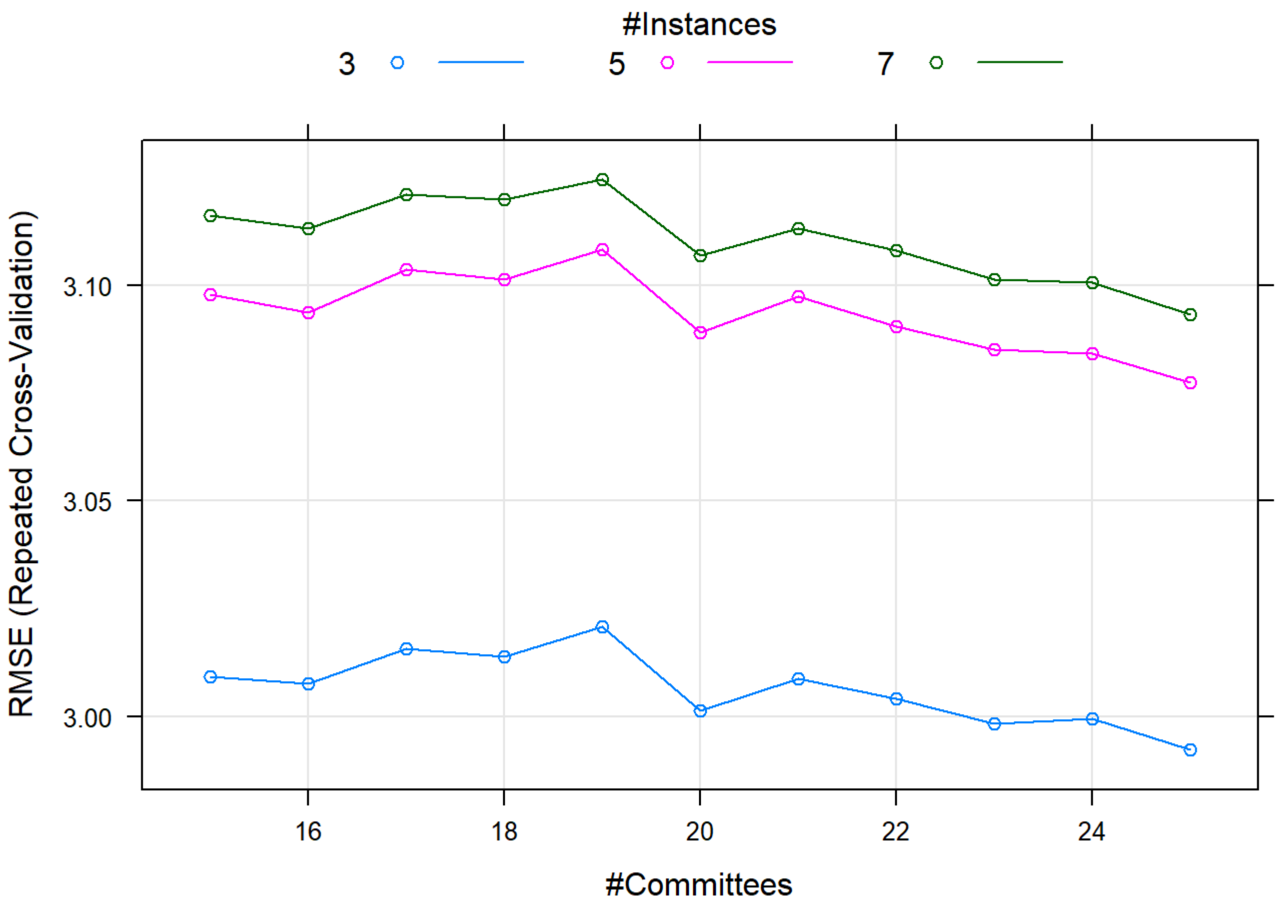
```
## Cubist
##
## 407 samples
##  13 predictor
##
## Pre-processing: Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 366, 367, 366, 367, 365, 367, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE      Rsquared   MAE
##    1          0          3.753432  0.8372372  2.468735
##    1          5          3.396743  0.8654549  2.207527
##    1          9          3.470867  0.8618118  2.237505
##   10          0          3.372730  0.8700258  2.233735
##   10          5          3.104315  0.8879016  1.978900
##   10          9          3.164513  0.8841651  2.012710
##   20          0          3.373670  0.8694293  2.228746
##   20          5          3.089250  0.8888146  1.965454
##   20          9          3.148450  0.8854011  1.992379
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were committees = 20 and neighbors = 5.
```

Let’s dive deeper into Cubist and see if we can tune it further and get more skill out of it.

```
# Tune the Cubist algorithm
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
set.seed(7)
grid <- expand.grid(.committees=seq(15, 25, by=1), .neighbors=c(3, 5, 7))
tune.cubist <- train(medv~., data=dataset, method="cubist", metric=metric, preProc=c("BoxCox"),
 tuneGrid=grid, trControl=control)
print(tune.cubist)
```

```
## Cubist
##
## 407 samples
##  13 predictor
##
## Pre-processing: Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 366, 367, 366, 367, 365, 367, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE      Rsquared   MAE
##   15          3          3.009082  0.8941074  1.919529
##   15          5          3.098053  0.8884253  1.976015
##   15          7          3.116386  0.8874419  1.979539
##   16          3          3.007537  0.8944762  1.919600
##   16          5          3.093709  0.8889310  1.975111
##   16          7          3.113242  0.8878123  1.978636
##   17          3          3.015645  0.8937711  1.923880
##   17          5          3.103834  0.8880055  1.977733
##   17          7          3.121269  0.8871594  1.982526
##   18          3          3.013869  0.8937863  1.921819
##   18          5          3.101501  0.8880896  1.973930
##   18          7          3.120030  0.8871029  1.978973
##   19          3          3.020826  0.8932599  1.921864
##   19          5          3.108476  0.8875403  1.974250
##   19          7          3.124609  0.8868355  1.978953
##   20          3          3.001396  0.8944892  1.911889
##   20          5          3.089250  0.8888146  1.965454
##   20          7          3.106933  0.8879802  1.969943
##   21          3          3.008654  0.8939703  1.913532
##   21          5          3.097445  0.8881274  1.968988
##   21          7          3.113314  0.8874920  1.973820
##   22          3          3.004181  0.8945197  1.913736
##   22          5          3.090638  0.8887730  1.966745
##   22          7          3.108176  0.8879460  1.971023
##   23          3          2.998382  0.8948426  1.910446
##   23          5          3.085185  0.8891198  1.964406
##   23          7          3.101469  0.8883363  1.969268
##   24          3          2.999325  0.8948105  1.912269
##   24          5          3.084216  0.8891824  1.964024
##   24          7          3.100808  0.8883295  1.968827
##   25          3          2.992166  0.8951772  1.907429
##   25          5          3.077461  0.8896248  1.959972
##   25          7          3.093274  0.8888028  1.963996
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were committees = 25 and neighbors = 3.
```

```
plot(tune.cubist)
```



We can see that the best RMSE was achieved with committees = 20 and neighbors = 5.

We can see that we have achieved a more accurate model again with an RMSE of 2.822 using committees = 18 and neighbors = 3.

8 Finalize Model

It looks like that cubist results in our most accurate model. Let's ???nalize it by creating a new standalone Cubist model with the parameters above trained using the whole dataset.

```
# prepare the data transform using training data
set.seed(7)
x <- dataset[,1:13]
y <- dataset[,14]
preprocessParams <- preProcess(x, method=c("BoxCox"))
trans_x <- predict(preprocessParams, x)
# train the final model
finalModel <- cubist(x=trans_x, y=y, committees=18)
summary(finalModel)
```

```
##
## Call:
## cubist.default(x = trans_x, y = y, committees = 18)
##
##
## Cubist [Release 2.07 GPL Edition]  Sun Apr 01 20:53:11 2018
## ---------------------------------
##
##     Target attribute `outcome'
##
## Read 407 cases (14 attributes) from undefined.data
##
## Model 1:
##
##   Rule 1/1: [84 cases, mean 13.75, range 5 to 25, est err 1.85]
##
##     if
##  nox > -0.497006
##     then
##  outcome = 28.8 - 3.48 lstat + 8.2 nox - 1.41 crim + 5.3 dis + 3e-005 b
##
##   Rule 1/2: [166 cases, mean 19.48, range 7 to 31, est err 2.05]
##
##     if
##  nox <= -0.497006
##  lstat > 2.858393
##     then
##  outcome = 158.68 - 2.35 lstat + 1.8 rad - 75 tax - 2.6 dis
##            - 0.037 ptratio + 10 rm - 0.0075 age + 2.3e-005 b + 1.6 chas
##
##   Rule 1/3: [107 cases, mean 25.59, range 18.6 to 37.2, est err 1.86]
##
##     if
##  rm <= 1.954587
##  dis > 0.5868974
##  lstat <= 2.858393
##     then
##  outcome = 17.94 + 49.3 rm - 4.3 dis - 1.71 lstat - 0.014 age
##            - 0.46 indus - 0.023 ptratio - 36 tax - 0.5 nox + 0.1 rad
##
##   Rule 1/4: [4 cases, mean 31.15, range 23.3 to 50, est err 1.69]
##
##     if
##  dis <= 0.5868974
##  b <= 66469.73
##  lstat <= 2.858393
##     then
##  outcome = 71.04 - 60.5 dis - 4.99 lstat
##
##   Rule 1/5: [9 cases, mean 38.21, range 17.8 to 50, est err 2.17]
##
##     if
##  rm > 1.954587
```

```
##  dis > 0.5868974
##  tax > 1.894297
##     then
##  outcome = 2234.56 - 1152 tax - 11.9 dis - 0.9 lstat + 7.6 rm
##            - 0.012 ptratio
##
##   Rule 1/6: [38 cases, mean 40.13, range 31 to 50, est err 2.55]
##
##     if
##  rm > 1.954587
##  tax <= 1.894297
##     then
##  outcome = 391.64 - 0.000497 b + 80.8 rm - 246 tax - 0.0294 age
##            - 0.047 ptratio - 1.52 lstat - 2.4 dis
##
##   Rule 1/7: [4 cases, mean 50.00, range 50 to 50, est err 0.00]
##
##     if
##  dis <= 0.5868974
##  b > 66469.73
##  lstat <= 2.858393
##     then
##  outcome = 50
##
## Model 2:
##
##   Rule 2/1: [10 cases, mean 7.92, range 5 to 12.3, est err 3.49]
##
##     if
##  nox > -0.4727541
##  dis <= 0.462979
##  ptratio > 149.145
##     then
##  outcome = 244.69 + 544.4 nox - 0.3 dis - 0.16 lstat
##
##   Rule 2/2: [9 cases, mean 12.56, range 10.2 to 15, est err 3.06]
##
##     if
##  nox <= -0.4727541
##  dis <= 0.462979
##  b > 67032.41
##  lstat > 1.931448
##     then
##  outcome = 31.3 - 0.48 lstat - 0.3 dis - 8 tax + 1.4 rm - 0.004 ptratio
##            - 0.06 indus
##
##   Rule 2/3: [145 cases, mean 18.41, range 7 to 35.2, est err 2.38]
##
##     if
##  dis > 0.462979
##  ptratio > 149.145
##  b <= 77263.3
##  lstat > 1.931448
##     then
```

```
##  outcome = 42.95 - 4.01 lstat - 0.042 ptratio + 5.7e-005 b + 5 rm
##            - 0.0055 age - 0.4 dis - 7 tax - 0.06 indus
##
##   Rule 2/4: [116 cases, mean 22.59, range 8.3 to 43.8, est err 2.25]
##
##     if
##  dis > 0.462979
##  tax > 1.857866
##  ptratio > 149.145
##  b > 77263.3
##     then
##  outcome = 120.83 - 0.001783 b + 36.4 rm - 0.026 age - 0.52 lstat
##            - 0.5 dis - 0.11 indus - 10 tax
##
##   Rule 2/5: [74 cases, mean 23.58, range 11.8 to 50, est err 2.37]
##
##     if
##  ptratio <= 149.145
##  lstat > 1.931448
##     then
##  outcome = -65.34 + 53.6 rm - 0.112 ptratio + 7.7e-005 b - 0.32 lstat
##            - 0.3 dis
##
##   Rule 2/6: [11 cases, mean 23.62, range 6.3 to 50, est err 7.87]
##
##     if
##  dis <= 0.462979
##  ptratio > 149.145
##  b <= 67032.41
##     then
##  outcome = 72.68 - 117.8 dis - 37.4 nox - 4.66 lstat - 0.000222 b
##
##   Rule 2/7: [11 cases, mean 24.03, range 15.7 to 39.8, est err 5.34]
##
##     if
##  tax <= 1.857866
##  lstat > 1.931448
##     then
##  outcome = 130.83 - 0.477 ptratio - 4.17 lstat - 0.0344 age - 0.2 dis
##
##   Rule 2/8: [9 cases, mean 28.52, range 22.5 to 50, est err 7.28]
##
##     if
##  rm <= 1.895669
##  lstat <= 1.931448
##     then
##  outcome = 77.55 - 33.16 lstat + 3.4 rm - 0.7 dis
##
##   Rule 2/9: [61 cases, mean 36.05, range 21 to 50, est err 3.01]
##
##     if
##  rm > 1.895669
##  tax <= 1.894297
##     then
```

```
##  outcome = 398.49 + 98.7 rm - 288 tax - 0.0433 age - 7.5 dis - 0.28 lstat
##
##   Rule 2/10: [13 cases, mean 42.53, range 30.5 to 50, est err 2.91]
##
##     if
##  tax > 1.894297
##  lstat <= 1.931448
##     then
##  outcome = -2.05 + 0.032 age + 21.8 rm - 3.2 dis
##
## Model 3:
##
##   Rule 3/1: [68 cases, mean 13.12, range 5 to 25, est err 2.21]
##
##     if
##  nox > -0.497006
##  ptratio > 109.02
##     then
##  outcome = 51.26 + 41.9 nox - 2.06 crim - 4.11 lstat
##
##   Rule 3/2: [171 cases, mean 19.94, range 7 to 50, est err 2.60]
##
##     if
##  nox <= -0.497006
##  rm <= 1.831781
##  lstat > 1.739722
##     then
##  outcome = 76.77 + 25.7 rm - 0.063 ptratio - 4.2 dis + 0.56 crim
##            - 1.15 lstat - 0.01 age - 42 tax - 0.37 indus + 3e-005 b
##            + 0.012 zn
##
##   Rule 3/3: [35 cases, mean 24.17, range 11.8 to 50, est err 3.79]
##
##     if
##  ptratio <= 109.02
##  lstat > 1.739722
##     then
##  outcome = 113.14 - 0.352 ptratio - 5.85 lstat - 1.3 dis + 5.3 rm
##            - 1.5 nox - 24 tax + 0.0024 age
##
##   Rule 3/4: [118 cases, mean 27.40, range 16.1 to 50, est err 2.09]
##
##     if
##  nox <= -0.497006
##  rm > 1.831781
##  lstat > 1.739722
##     then
##  outcome = 67.96 + 56.8 rm - 0.05 ptratio - 1.89 lstat - 71 tax - 1.3 dis
##            + 0.24 crim - 0.16 indus - 0.0032 age + 1.3e-005 b + 0.005 zn
##
##   Rule 3/5: [25 cases, mean 38.41, range 24.8 to 50, est err 3.51]
##
##     if
##  dis > 1.162901
```

```
##  lstat <= 1.739722
##     then
##  outcome = 283.78 + 96.9 rm - 0.0574 age - 219 tax - 7.6 dis
##            - 0.067 ptratio
##
##   Rule 3/6: [9 cases, mean 49.42, range 44.8 to 50, est err 3.35]
##
##     if
##  dis <= 1.162901
##  lstat <= 1.739722
##     then
##  outcome = 56.35 - 9 dis
##
## Model 4:
##
##   Rule 4/1: [90 cases, mean 13.70, range 5 to 27.9, est err 3.46]
##
##     if
##  dis <= 0.7244036
##  lstat > 2.858393
##     then
##  outcome = 203.21 - 19.6 dis - 7.53 lstat + 0.0679 age - 6.3 nox - 80 tax
##            - 5.1e-005 b - 8 rm
##
##   Rule 4/2: [247 cases, mean 17.48, range 5 to 31, est err 2.91]
##
##     if
##  lstat > 2.858393
##     then
##  outcome = -2.29 + 21.8 rm - 0.0239 age - 2.18 lstat - 0.034 ptratio
##            + 5.2e-005 b + 2.5 nox - 0.1 crim - 0.3 dis
##
##   Rule 4/3: [41 cases, mean 20.49, range 14.4 to 29.6, est err 2.58]
##
##     if
##  nox <= -1.04499
##  lstat > 2.858393
##     then
##  outcome = 62.87 - 0.000266 b + 10.1 nox - 5.9 dis
##
##   Rule 4/4: [103 cases, mean 25.88, range 18.6 to 50, est err 2.66]
##
##     if
##  rm <= 1.937158
##  lstat <= 2.858393
##     then
##  outcome = -38.51 + 52.8 rm - 4.7 dis - 1.39 lstat - 0.012 age - 0.8 nox
##            + 0.12 crim - 12 tax - 0.006 ptratio - 0.09 indus + 5e-006 b
##
##   Rule 4/5: [47 cases, mean 38.35, range 23.6 to 50, est err 3.20]
##
##     if
##  rm > 1.937158
##  tax <= 1.896025
```

```
##     then
##  outcome = 697.02 - 0.000827 b + 102.5 rm - 418 tax - 0.0401 age
##            - 4.8 dis - 0.038 ptratio
##
##   Rule 4/6: [11 cases, mean 38.45, range 21.9 to 50, est err 6.32]
##
##     if
##  rm > 1.937158
##  dis > 0.5868974
##  tax > 1.896025
##  lstat <= 2.858393
##     then
##  outcome = -78.2 + 65 rm - 0.087 ptratio
##
##   Rule 4/7: [8 cases, mean 40.58, range 23.3 to 50, est err 22.46]
##
##     if
##  dis <= 0.5868974
##  lstat <= 2.858393
##     then
##  outcome = 58.89 - 104.3 dis + 11.72 lstat
##
## Model 5:
##
##   Rule 5/1: [84 cases, mean 13.75, range 5 to 25, est err 3.17]
##
##     if
##  nox > -0.497006
##     then
##  outcome = 40.91 + 12.2 dis - 2.6 crim + 8.3 nox - 2.85 lstat - 10.2 rm
##            + 3e-005 b
##
##   Rule 5/2: [12 cases, mean 15.95, range 13.2 to 23.7, est err 3.47]
##
##     if
##  nox <= -0.497006
##  lstat > 4.439301
##     then
##  outcome = 17.92
##
##   Rule 5/3: [156 cases, mean 19.81, range 10.2 to 29.6, est err 1.91]
##
##     if
##  nox <= -0.497006
##  rm <= 1.831301
##  dis > 0.5626968
##     then
##  outcome = 139.74 + 23.4 rm - 79 tax + 1.8 rad - 0.041 ptratio - 2.8 dis
##            - 1.24 lstat - 0.0069 age + 2.3e-005 b + 1.7 chas + 0.016 zn
##
##   Rule 5/4: [124 cases, mean 23.08, range 7.2 to 50, est err 3.41]
##
##     if
##  rm > 1.831301
```

```
##  lstat > 1.987397
##     then
##  outcome = 102.95 + 43.6 rm - 0.053 ptratio - 79 tax - 1.02 lstat
##            - 1.1 dis + 0.5 rad - 0.9 nox + 0.0019 age + 0.008 zn
##
##   Rule 5/5: [10 cases, mean 26.73, range 7 to 50, est err 9.26]
##
##     if
##  nox <= -0.497006
##  rm <= 1.831301
##  dis <= 0.5626968
##  lstat <= 4.439301
##     then
##  outcome = 116.88 - 24.41 lstat - 3.1 dis
##
##   Rule 5/6: [62 cases, mean 36.86, range 21.9 to 50, est err 4.83]
##
##     if
##  lstat <= 1.987397
##     then
##  outcome = -111.16 + 78.2 rm - 6.3 dis - 1.52 crim - 0.43 lstat
##
##   Rule 5/7: [13 cases, mean 43.86, range 21.9 to 50, est err 9.72]
##
##     if
##  age > 229.474
##  lstat <= 1.987397
##     then
##  outcome = -60.3 + 0.4787 age - 1.09 lstat - 1 dis - 1.2 nox - 14 tax
##            - 0.006 ptratio + 1.6 rm + 0.2 rad
##
## Model 6:
##
##   Rule 6/1: [29 cases, mean 13.33, range 7 to 27.5, est err 4.31]
##
##     if
##  b <= 16084.5
##     then
##  outcome = 19.31 + 0.000779 b - 0.38 lstat - 0.4 dis + 1 rm
##            - 0.003 ptratio - 5 tax
##
##   Rule 6/2: [247 cases, mean 17.48, range 5 to 31, est err 2.54]
##
##     if
##  lstat > 2.858393
##     then
##  outcome = 67.94 - 3.35 lstat - 0.68 crim - 0.0142 age + 3.3 nox
##            - 0.015 ptratio + 2.6e-005 b + 0.4 rad - 17 tax
##
##   Rule 6/3: [9 cases, mean 21.33, range 15.7 to 29.6, est err 3.41]
##
##     if
##  tax <= 1.857866
##  lstat > 2.858393
```

```
##     then
##  outcome = 4.61 + 32.7 dis - 0.36 lstat - 0.003 ptratio + 0.9 rm - 5 tax
##
##   Rule 6/4: [114 cases, mean 28.38, range 18.6 to 50, est err 2.17]
##
##     if
##  dis > 1.230868
##  lstat <= 2.858393
##     then
##  outcome = -84.69 + 73.9 rm - 5.8 dis - 0.0285 age - 0.84 indus
##            - 0.64 lstat + 0.13 crim - 0.7 nox - 0.004 ptratio - 6 tax
##
##   Rule 6/5: [29 cases, mean 33.81, range 20.6 to 50, est err 3.83]
##
##     if
##  dis <= 1.230868
##  b > 73935.01
##  lstat <= 2.858393
##     then
##  outcome = -82.01 - 14.9 dis + 69.6 rm + 0.92 crim - 1.6 lstat - 3.2 nox
##            - 0.48 indus - 0.0095 age
##
##   Rule 6/6: [12 cases, mean 39.08, range 21.9 to 50, est err 8.85]
##
##     if
##  dis <= 0.6604174
##  lstat <= 2.858393
##     then
##  outcome = -5.71 - 62.2 dis + 0.001034 b - 0.21 lstat
##
##   Rule 6/7: [8 cases, mean 41.85, range 25 to 50, est err 11.11]
##
##     if
##  dis > 0.6604174
##  dis <= 1.230868
##  b <= 73935.01
##  lstat <= 2.858393
##     then
##  outcome = -98.17 + 74.9 rm - 11.2 dis + 0.000142 b + 0.68 crim
##            - 1.17 lstat - 2.3 nox - 0.35 indus - 0.0069 age
##
## Model 7:
##
##   Rule 7/1: [84 cases, mean 13.75, range 5 to 25, est err 2.49]
##
##     if
##  nox > -0.497006
##     then
##  outcome = 25.08 + 11 dis - 2.31 crim - 3.33 lstat + 6.9 nox + 2.8e-005 b
##
##   Rule 7/2: [59 cases, mean 14.73, range 5 to 50, est err 7.01]
##
##     if
##  dis <= 0.5626968
```

```
##  lstat > 1.931448
##     then
##  outcome = 16.83 + 0.06 crim + 1.1 rm - 0.003 ptratio - 0.2 dis
##
##   Rule 7/3: [236 cases, mean 21.99, range 10.2 to 50, est err 2.22]
##
##     if
##  nox <= -0.497006
##  dis > 0.5626968
##  tax > 1.865769
##  lstat > 1.931448
##     then
##  outcome = -18.15 + 35.4 rm - 0.0248 age - 0.045 ptratio + 0.58 crim
##            - 2.2 dis + 5.8e-005 b + 1.2 lstat - 1 nox - 9 tax
##
##   Rule 7/4: [10 cases, mean 25.41, range 7 to 50, est err 12.49]
##
##     if
##  nox <= -0.497006
##  dis <= 0.5626968
##  b <= 67032.41
##     then
##  outcome = 81.16 - 115.7 dis - 0.000217 b - 0.49 lstat
##
##   Rule 7/5: [58 cases, mean 25.55, range 16.5 to 50, est err 2.33]
##
##     if
##  nox <= -0.497006
##  ptratio <= 149.145
##  lstat > 1.931448
##     then
##  outcome = -22.5 + 47.6 rm - 0.089 ptratio + 3.1 nox - 0.66 lstat
##            - 0.6 dis - 12 tax + 0.0019 age + 0.08 crim
##
##   Rule 7/6: [20 cases, mean 29.26, range 15.7 to 50, est err 4.96]
##
##     if
##  tax <= 1.865769
##  ptratio > 149.145
##     then
##  outcome = 53.7 - 0.394 ptratio + 12 dis + 17.6 nox + 2.64 crim + 33.6 rm
##            - 2.74 lstat
##
##   Rule 7/7: [6 cases, mean 29.48, range 22.5 to 50, est err 6.03]
##
##     if
##  rm <= 1.882057
##  lstat <= 1.931448
##     then
##  outcome = 220.69 - 106 rm
##
##   Rule 7/8: [37 cases, mean 38.17, range 22.8 to 50, est err 3.89]
##
##     if
```

```
##  rm > 1.882057
##  tax <= 1.894297
##  lstat <= 1.931448
##     then
##  outcome = 767.63 + 111.8 rm - 506 tax - 0.0255 age - 0.33 lstat
##            - 0.5 nox - 0.3 dis - 0.003 ptratio
##
##   Rule 7/9: [12 cases, mean 41.91, range 30.5 to 50, est err 2.56]
##
##     if
##  rm > 1.882057
##  tax > 1.894297
##  lstat <= 1.931448
##     then
##  outcome = 46.61 + 0.0476 age + 10.8 rm - 1.11 lstat - 1.6 nox - 1 dis
##            - 0.009 ptratio - 16 tax + 0.1 crim
##
## Model 8:
##
##   Rule 8/1: [39 cases, mean 12.55, range 5 to 27.9, est err 3.96]
##
##     if
##  crim > 2.382708
##     then
##  outcome = 54.82 - 5.35 crim - 6.25 lstat
##
##   Rule 8/2: [141 cases, mean 17.25, range 6.3 to 31, est err 2.50]
##
##     if
##  crim <= 2.382708
##  nox > -0.8796992
##  lstat > 2.858393
##     then
##  outcome = 60.02 - 5.63 lstat + 7.7 nox - 0.97 crim + 1.4 dis
##            - 0.0078 age + 3.2 rm + 1.7e-005 b - 0.008 ptratio - 12 tax
##
##   Rule 8/3: [67 cases, mean 20.84, range 14.4 to 29.6, est err 2.09]
##
##     if
##  nox <= -0.8796992
##  lstat > 2.858393
##     then
##  outcome = 48.44 + 8.1 nox - 1.47 lstat + 2.6 rm - 0.5 dis
##            - 0.007 ptratio - 0.11 crim - 9 tax
##
##   Rule 8/4: [160 cases, mean 30.52, range 18.6 to 50, est err 3.26]
##
##     if
##  lstat <= 2.858393
##     then
##  outcome = -31.85 + 60.1 rm - 6.3 dis - 2.81 lstat - 0.0153 age
##            + 0.27 crim - 18 tax + 0.015 zn - 0.009 ptratio
##
##   Rule 8/5: [8 cases, mean 40.58, range 23.3 to 50, est err 9.14]
```

```
##
##     if
##  dis <= 0.5868974
##  lstat <= 2.858393
##     then
##  outcome = 75.2 - 87.2 dis
##
## Model 9:
##
##   Rule 9/1: [83 cases, mean 13.65, range 5 to 25, est err 2.25]
##
##     if
##  nox > -0.497006
##  lstat > 2.150069
##     then
##  outcome = 22.54 + 11.7 dis + 10.8 nox - 1.57 crim + 6.3e-005 b
##            - 0.15 lstat - 6 tax
##
##   Rule 9/2: [29 cases, mean 19.16, range 7 to 50, est err 7.45]
##
##     if
##  nox <= -0.497006
##  rm <= 1.85661
##  dis <= 0.7285143
##  lstat > 2.150069
##     then
##  outcome = 387.41 - 44.9 dis + 2.15 crim - 4.7 lstat - 199 tax + 32 rm
##
##   Rule 9/3: [167 cases, mean 20.49, range 12.7 to 29.6, est err 2.00]
##
##     if
##  nox <= -0.497006
##  rm <= 1.85661
##  dis > 0.7285143
##     then
##  outcome = 132.11 + 25.2 rm - 0.057 ptratio - 0.0213 age - 3.3 dis
##            - 75 tax + 1.4 rad + 0.49 crim + 0.21 indus + 0.44 lstat
##
##   Rule 9/4: [60 cases, mean 26.86, range 16.1 to 50, est err 2.12]
##
##     if
##  nox <= -0.497006
##  rm > 1.85661
##  lstat > 2.150069
##     then
##  outcome = 117.46 + 69.1 rm - 113 tax - 0.053 ptratio - 0.7 dis
##            - 0.0034 age + 0.14 crim + 0.3 rad
##
##   Rule 9/5: [76 cases, mean 35.06, range 20.6 to 50, est err 5.99]
##
##     if
##  lstat <= 2.150069
##     then
##  outcome = 147.59 + 35.6 rm - 93 tax - 0.046 ptratio - 1.41 lstat
```

```
##            + 2.2 nox - 1.4 dis + 0.033 zn + 0.5 rad + 0.0031 age
##            + 0.6 chas
##
##   Rule 9/6: [26 cases, mean 38.57, range 23.6 to 50, est err 4.54]
##
##     if
##  nox <= -0.6286645
##  rm > 1.914272
##  tax > 1.877141
##  lstat <= 2.150069
##     then
##  outcome = -1249.52 + 111.9 rm + 580 tax + 20.9 nox - 0.0495 age
##            - 1.29 lstat - 0.014 ptratio + 0.005 zn
##
##   Rule 9/7: [19 cases, mean 38.87, range 28.5 to 50, est err 4.77]
##
##     if
##  nox <= -0.6286645
##  rm > 1.914272
##  tax <= 1.877141
##  lstat <= 2.150069
##     then
##  outcome = -191.22 + 159.9 rm + 3.92 lstat + 2.7 nox - 45 tax
##            - 0.022 ptratio + 0.019 zn
##
##   Rule 9/8: [6 cases, mean 45.12, range 21.9 to 50, est err 25.03]
##
##     if
##  nox > -0.6286645
##  lstat <= 2.150069
##     then
##  outcome = -119.53 - 313.9 nox - 9.3 chas
##
## Model 10:
##
##   Rule 10/1: [221 cases, mean 18.34, range 5 to 50, est err 3.01]
##
##     if
##  rm <= 1.831301
##  lstat > 1.931448
##     then
##  outcome = 98.79 - 4.59 lstat - 0.76 crim + 10 rm - 1.9 dis - 41 tax
##            - 0.36 indus + 0.8 rad - 0.017 ptratio
##
##   Rule 10/2: [185 cases, mean 27.55, range 7.2 to 50, est err 3.26]
##
##     if
##  rm > 1.831301
##     then
##  outcome = 143.21 - 6.01 lstat + 34.3 rm - 3.5 dis - 84 tax - 0.33 indus
##            - 0.016 ptratio - 0.11 crim + 0.2 rad
##
##   Rule 10/3: [9 cases, mean 28.52, range 22.5 to 50, est err 5.73]
##
```

```
##     if
##  rm <= 1.895669
##  lstat <= 1.931448
##     then
##  outcome = 201.43 - 91.7 rm - 2.4 dis
##
##   Rule 10/4: [40 cases, mean 35.86, range 22.5 to 50, est err 3.20]
##
##     if
##  crim <= -1.051538
##  lstat <= 1.931448
##     then
##  outcome = -136.13 + 96.1 rm - 7.9 dis - 0.0206 age
##
##   Rule 10/5: [12 cases, mean 46.13, range 31.5 to 50, est err 8.21]
##
##     if
##  crim > -1.051538
##  rm > 1.895669
##  lstat <= 1.931448
##     then
##  outcome = 4.13 + 3.95 crim - 7.6 dis + 28.7 rm - 0.0197 age
##
## Model 11:
##
##   Rule 11/1: [84 cases, mean 13.75, range 5 to 25, est err 2.87]
##
##     if
##  nox > -0.497006
##     then
##  outcome = 41.07 + 15.3 dis + 13.5 nox - 2.3 crim - 1.64 lstat - 13.3 rm
##            + 6.1e-005 b
##
##   Rule 11/2: [169 cases, mean 19.61, range 7 to 33.8, est err 3.09]
##
##     if
##  nox <= -0.497006
##  lstat > 2.848535
##     then
##  outcome = 36.73 - 0.069 ptratio + 0.84 crim - 3.2 dis - 0.0122 age
##            + 4.8e-005 b - 0.33 lstat - 0.4 nox + 1 rm
##
##   Rule 11/3: [157 cases, mean 30.59, range 18.6 to 50, est err 4.09]
##
##     if
##  lstat <= 2.848535
##     then
##  outcome = 54.82 + 87.9 rm - 0.026 age - 0.062 ptratio + 1.01 crim
##            - 91 tax + 4.8 nox - 1.1 dis
##
##   Rule 11/4: [11 cases, mean 39.32, range 21.9 to 50, est err 28.68]
##
##     if
##  dis <= 0.6492998
```

```
##  lstat <= 2.848535
##     then
##  outcome = 58.77 - 179.7 dis - 16.74 crim + 9.48 lstat + 31.1 rm
##
## Model 12:
##
##   Rule 12/1: [300 cases, mean 19.06, range 5 to 50, est err 2.96]
##
##     if
##  rm <= 1.887978
##  lstat > 1.805082
##     then
##  outcome = 118.53 - 5.23 lstat + 13.1 rm - 0.59 indus - 2.2 dis - 53 tax
##
##   Rule 12/2: [67 cases, mean 28.08, range 7.5 to 50, est err 3.89]
##
##     if
##  rm > 1.887978
##  lstat > 1.805082
##     then
##  outcome = 243.66 + 32.5 rm - 143 tax - 4.5 dis - 2.36 lstat - 5 nox
##            - 0.53 indus
##
##   Rule 12/3: [31 cases, mean 38.23, range 22.8 to 50, est err 3.37]
##
##     if
##  tax <= 1.899749
##  lstat <= 1.805082
##     then
##  outcome = -126.48 + 82.5 rm - 3.09 crim - 8.4e-005 b - 0.81 lstat
##
##   Rule 12/4: [9 cases, mean 46.42, range 32.9 to 50, est err 8.91]
##
##     if
##  tax > 1.899749
##  lstat <= 1.805082
##     then
##  outcome = 53.55
##
## Model 13:
##
##   Rule 13/1: [84 cases, mean 13.75, range 5 to 25, est err 3.29]
##
##     if
##  nox > -0.497006
##     then
##  outcome = 36.89 + 16.5 dis - 3.04 crim + 15.4 nox - 14.3 rm + 7.1e-005 b
##
##   Rule 13/2: [169 cases, mean 19.61, range 7 to 33.8, est err 3.36]
##
##     if
##  nox <= -0.497006
##  lstat > 2.848535
##     then
```

```
##  outcome = 215.56 + 3.5 rad - 0.074 ptratio - 98 tax - 3.4 dis
##            - 0.0119 age + 5.5e-005 b + 0.33 indus
##
##   Rule 13/3: [298 cases, mean 24.85, range 7 to 50, est err 3.84]
##
##     if
##  crim <= 0.9690653
##     then
##  outcome = -128.58 + 90.9 rm - 5.5 dis - 0.0176 age + 0.77 crim
##            - 0.028 ptratio
##
##   Rule 13/4: [135 cases, mean 29.21, range 18.6 to 50, est err 2.85]
##
##     if
##  dis > 0.9708925
##  lstat <= 2.848535
##     then
##  outcome = 159.83 + 79.8 rm + 1.65 crim - 0.0294 age - 138 tax - 4.7 dis
##            - 0.058 ptratio
##
##   Rule 13/5: [9 cases, mean 38.40, range 21.9 to 50, est err 11.82]
##
##     if
##  crim > 0.9690653
##  lstat <= 2.848535
##     then
##  outcome = -9.01 - 41.5 dis + 40.7 rm
##
## Model 14:
##
##   Rule 14/1: [218 cases, mean 18.28, range 5 to 50, est err 3.15]
##
##     if
##  rm <= 1.831301
##  lstat > 2.150069
##     then
##  outcome = 92.35 - 4.33 lstat - 0.7 crim - 35 tax + 5 rm - 0.23 indus
##
##   Rule 14/2: [112 cases, mean 22.42, range 7.2 to 50, est err 2.59]
##
##     if
##  rm > 1.831301
##  tax > 1.857866
##  lstat > 2.150069
##     then
##  outcome = 34.48 - 5.61 lstat + 40.2 rm - 36 tax - 0.33 indus - 0.9 dis
##
##   Rule 14/3: [10 cases, mean 23.18, range 15.7 to 39.8, est err 5.98]
##
##     if
##  tax <= 1.857866
##  lstat > 2.150069
##     then
##  outcome = 32.05 + 30.2 dis - 4.58 lstat + 1.8 rm - 10 tax - 0.09 indus
```

```
##
##   Rule 14/4: [8 cases, mean 26.65, range 20.6 to 50, est err 6.28]
##
##     if
##  rm <= 1.849714
##  lstat <= 2.150069
##     then
##  outcome = 260.66 - 127.5 rm - 0.56 lstat - 0.3 dis - 0.07 crim
##
##   Rule 14/5: [68 cases, mean 36.05, range 21.9 to 50, est err 5.20]
##
##     if
##  rm > 1.849714
##  lstat <= 2.150069
##     then
##  outcome = -143.78 + 95.9 rm - 2.43 crim - 6.6 dis - 0.94 lstat
##            - 0.008 ptratio
##
##   Rule 14/6: [20 cases, mean 40.28, range 21.9 to 50, est err 9.15]
##
##     if
##  rm > 1.849714
##  age > 195.4278
##  lstat <= 2.150069
##     then
##  outcome = 41.37 + 0.1192 age - 10.24 lstat - 0.162 ptratio + 3.1 rm
##
## Model 15:
##
##   Rule 15/1: [84 cases, mean 13.75, range 5 to 25, est err 2.84]
##
##     if
##  nox > -0.497006
##     then
##  outcome = 41.69 + 13.1 dis + 15.4 nox - 2.35 crim - 2 lstat - 11.9 rm
##            + 6e-005 b
##
##   Rule 15/2: [166 cases, mean 19.48, range 7 to 31, est err 2.83]
##
##     if
##  nox <= -0.497006
##  lstat > 2.858393
##     then
##  outcome = 47.22 - 0.081 ptratio - 4.8 dis + 0.99 crim + 12.7 rm
##            - 0.0128 age + 5.1e-005 b - 0.67 lstat - 1 nox + 0.4 rad
##            - 15 tax
##
##   Rule 15/3: [160 cases, mean 30.52, range 18.6 to 50, est err 4.39]
##
##     if
##  lstat <= 2.858393
##     then
##  outcome = 85.06 + 81 rm + 1.46 crim - 4.8 dis - 0.0258 age
##            - 0.065 ptratio - 100 tax - 0.5 nox - 0.15 lstat + 0.1 rad
```

```
##
##   Rule 15/4: [11 cases, mean 39.32, range 21.9 to 50, est err 18.74]
##
##     if
##  dis <= 0.6492998
##  lstat <= 2.858393
##     then
##  outcome = 132.31 - 123.9 dis - 5.64 indus
##
## Model 16:
##
##   Rule 16/1: [62 cases, mean 16.43, range 5 to 50, est err 4.36]
##
##     if
##  dis <= 0.5626968
##     then
##  outcome = 71.14 - 35.8 dis - 10.27 lstat + 1 rm
##
##   Rule 16/2: [345 cases, mean 23.72, range 7.2 to 50, est err 3.33]
##
##     if
##  dis > 0.5626968
##     then
##  outcome = 67.61 + 33.1 rm - 2.21 lstat - 0.65 crim + 3.4 nox - 0.5 indus
##            - 0.0108 age - 51 tax + 4.9e-005 b
##
##   Rule 16/3: [15 cases, mean 29.29, range 15.7 to 50, est err 3.14]
##
##     if
##  tax <= 1.857866
##     then
##  outcome = 144.35 - 5.98 lstat + 18.4 rm - 0.78 indus - 2.8 dis - 67 tax
##
##   Rule 16/4: [6 cases, mean 30.28, range 22.8 to 50, est err 15.17]
##
##     if
##  rm <= 1.895669
##  lstat <= 1.805082
##     then
##  outcome = 294.96 - 134.8 rm - 0.000128 b - 0.96 lstat
##
##   Rule 16/5: [33 cases, mean 37.96, range 22.8 to 50, est err 4.98]
##
##     if
##  tax <= 1.900249
##  lstat <= 1.805082
##     then
##  outcome = -137.4 + 91.7 rm - 3.85 crim - 0.000181 b - 0.77 lstat
##
##   Rule 16/6: [6 cases, mean 50.00, range 50 to 50, est err 12.43]
##
##     if
##  rm > 1.895669
##  tax > 1.900249
```

```
##  lstat <= 1.805082
##     then
##  outcome = 1473.45 - 649 tax - 87.1 rm - 7.3e-005 b - 0.31 lstat
##
## Model 17:
##
##   Rule 17/1: [84 cases, mean 13.75, range 5 to 25, est err 2.83]
##
##     if
##  nox > -0.497006
##     then
##  outcome = 52.92 + 14.8 dis - 3.7 crim - 2.25 lstat - 17.7 rm
##
##   Rule 17/2: [9 cases, mean 13.79, range 7.5 to 21.9, est err 9.65]
##
##     if
##  nox > -0.5267175
##  rm > 1.898219
##     then
##  outcome = 19.26 - 5.03 crim
##
##   Rule 17/3: [243 cases, mean 21.17, range 7 to 50, est err 2.94]
##
##     if
##  nox <= -0.497006
##  rm <= 1.898219
##     then
##  outcome = 62.1 - 0.074 ptratio - 2.55 lstat - 4.2 dis + 1.8 rad
##            - 0.9 nox + 2.2 rm - 12 tax + 0.0017 age
##
##   Rule 17/4: [47 cases, mean 34.92, range 22 to 50, est err 5.09]
##
##     if
##  nox <= -0.5267175
##  rm > 1.898219
##  tax > 1.877141
##     then
##  outcome = -63.35 - 0.1167 age + 94.1 rm - 19.2 dis + 23.1 nox
##            - 0.111 ptratio - 0.28 lstat
##
##   Rule 17/5: [33 cases, mean 38.21, range 26.6 to 50, est err 4.06]
##
##     if
##  rm > 1.898219
##  tax <= 1.877141
##     then
##  outcome = -195.9 + 126.5 rm - 4.9 dis - 1.69 lstat - 0.0141 age
##            - 0.028 ptratio
##
## Model 18:
##
##   Rule 18/1: [250 cases, mean 17.59, range 5 to 33.8, est err 2.85]
##
##     if
```

```
##  lstat > 2.848535
##     then
##  outcome = 176.49 - 2.81 lstat - 92 tax + 15.1 rm - 0.0132 age
##            + 5.8e-005 b + 1 nox - 0.3 dis + 0.05 crim - 0.04 indus
##
##   Rule 18/2: [157 cases, mean 30.59, range 18.6 to 50, est err 4.38]
##
##     if
##  lstat <= 2.848535
##     then
##  outcome = 120.3 + 45 rm - 3.67 lstat - 91 tax - 3.2 nox - 0.6 dis
##            - 0.08 indus + 6e-006 b + 0.05 crim
##
##   Rule 18/3: [8 cases, mean 40.58, range 23.3 to 50, est err 18.43]
##
##     if
##  dis <= 0.5868974
##  lstat <= 2.848535
##     then
##  outcome = 90.38 - 114.5 dis
##
##
## Evaluation on training data (407 cases):
##
##     Average  |error|               1.72
##     Relative |error|               0.26
##     Correlation coefficient        0.97
##
##
##  Attribute usage:
##    Conds  Model
##
##     69%    85%    lstat
##     37%    52%    nox
##     35%    88%    rm
##     24%    87%    dis
##     12%    73%    tax
##      6%    66%    crim
##      6%    63%    ptratio
##      4%    50%    b
##           65%    age
##           38%    indus
##           27%    rad
##           10%    zn
##            5%    chas
##
##
## Time: 0.2 secs
```

We can now use this model to evaluate our held out validation dataset. Again, we must prepare the input data using the same Box-Cox transform.

```
# transform the validation dataset
set.seed(7)
val_x <- validation[,1:13]
trans_val_x <- predict(preprocessParams, val_x)
val_y <- validation[,14]
# use final model to make predictions on the validation dataset
predictions <- predict(finalModel, newdata=trans_val_x, neighbors=3)
# calculate RMSE
rmse <- RMSE(predictions, val_y)
r2 <- R2(predictions, val_y)
print(rmse)
```

```
## [1] 2.666336
```

We can see that the estimated RMSE on this unseen data is 2.666, lower but not too dissimilar from our expected RMSE of 2.822.