Data Science Project - Predictive Model

by MP

Dec 2019

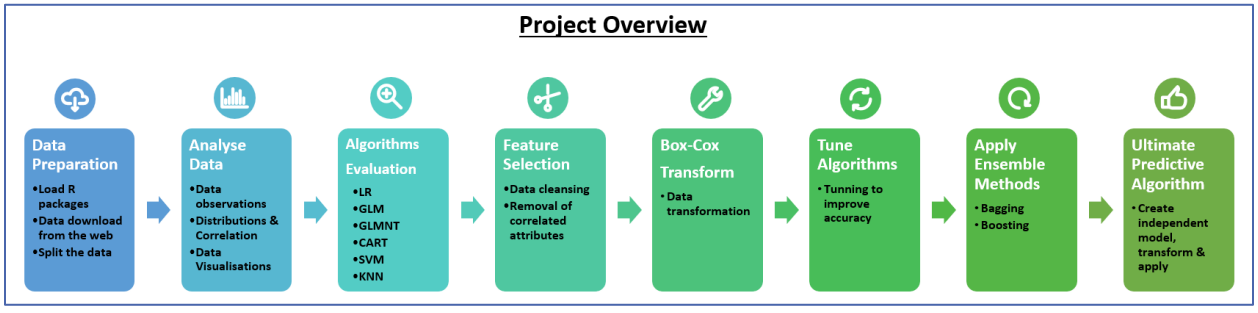
1 INTRODUCTION

In this project we look at the Boston House Price dataset to work through a regression predictive model for predicting the housing prices in Boston. We first analyse the data and then try to predict Boston house value with a model. In order to refine the accuracy of the model, we will transform the data, use algorithms to tune data and apply ensemble methods of bagging and boosting.

1.1 PROJECT OVERVIEW

Below is a quick overview of the project.

```
# Project Overview Visual
knitr::include_graphics("https://github.com/Matrix0007/datasets/blob/master/Project_Overview.png?raw=true")
```



2 LOAD DATA PACKAGES AND DATA

We need to load the pakages as listed below;

```
#load packages if required
if(!require(mlbench)) install.packages("mlbench", repos = "http://cran.us.r-project.o
rg")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(lattice)) install.packages("lattice", repos = "http://cran.us.r-project.o
rg")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.o
rg")
if(!require(Cubist)) install.packages("Cubist", repos = "http://cran.us.r-project.org
")
if(!require(jpeg)) install.packages("jpeg", repos = "http://cran.us.r-project.org")
if(!require(readr)) install.packages("readr", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-proje
ct.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")

library(mlbench)
library(caret)
library(corrplot)
library(tibble)
library(Cubist)
library(jpeg)
library(readr)
library(tidyverse)
library(knitr)
```

Once the packages are loaded, we download the data from the web by using the below code;

```
#load data from url
dta <- url("http://course1.winona.edu/bdeppa/Stat%20425/Data/Boston_Housing.csv")
TBdta <- as.data.frame(read.csv(dta, check.names = FALSE))
```

3 VALIDATION DATASET

It is a good idea to use a validation hold out set. This is a sample of the data that we hold back from our analysis and modeling. We use it right at the end of our project to confirm the accuracy of our final model. It is a smoke test that we can use to see if we messed up and to give us confidence on our estimates of accuracy on unseen data.

```
# Split out validation dataset
# create a list of 80% of the rows in the original dataset we can use for training
set.seed(7)
validateIndex <- caret::createDataPartition(TBdta$MEDV, p=0.80, list=FALSE)
# select 20% of the data for validation
validateSet <- TBdta[-validateIndex,]
# use the remaining 80% of data to training and testing the models
DTset <- TBdta[validateIndex,]
```

4 ANALYSE DATA

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

4.1 DATA OBSERVATIONS

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 defined as Follows;

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

4.1.1 Dimensions and the class

```
# check dimensions of DTset
dim(DTset)

## [1] 407  14

# review attribute types
sapply(DTset, class)

##      CRIM        ZN     INDUS      CHAS       NOX        RM       AGE
## "numeric" "numeric" "numeric" "integer" "numeric" "numeric" "numeric"
##       DIS       RAD       TAX   PTRATIO         B     LSTAT      MEDV
## "numeric" "integer" "integer" "numeric" "numeric" "numeric" "numeric"
```

The above results show that;
• There are 407 instances and 14 attributes
• We also note the attributes as “numeric” and “integer”

4.1.2 Snapshot of data

To understand the data better we review the first 10 rows of data.

```
# data first 10 rows
head(DTset, n=10)

##       CRIM   ZN INDUS CHAS   NOX    RM  AGE    DIS RAD TAX PTRATIO      B
## 1  0.00632 18.0  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90
## 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
## 10 0.17004 12.5  7.87    0 0.524 6.004 85.9 6.5921   5 311    15.2 386.71
## 11 0.22489 12.5  7.87    0 0.524 6.377 94.3 6.3467   5 311    15.2 392.52
## 12 0.11747 12.5  7.87    0 0.524 6.009 82.9 6.2267   5 311    15.2 396.90
```

```
##    LSTAT MEDV
## 1   4.98 24.0
## 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
## 10 17.10 18.9
## 11 20.45 15.0
## 12 13.27 18.9
```

We note that the scale range for the attributes are very broad due to the varied unit.
Transformation of these can assist us.

4.1.3 Distribution of the attributes

Below is a summary of attributes distribution.

```
# summary of the attributes
summary(DTset)
```

```
##       CRIM                ZN              INDUS            CHAS
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.74   Min.   :0.00000
##  1st Qu.: 0.07964   1st Qu.:  0.00   1st Qu.: 4.93   1st Qu.:0.00000
##  Median : 0.26838   Median :  0.00   Median : 8.56   Median :0.00000
##  Mean   : 3.63725   Mean   : 11.92   Mean   :11.00   Mean   :0.07125
##  3rd Qu.: 3.68567   3rd Qu.: 15.00   3rd Qu.:18.10   3rd Qu.:0.00000
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000
##       NOX               RM              AGE              DIS
##  Min.   :0.3850   Min.   :3.561   Min.   :  6.20   Min.   : 1.130
##  1st Qu.:0.4480   1st Qu.:5.888   1st Qu.: 42.70   1st Qu.: 2.109
##  Median :0.5380   Median :6.212   Median : 77.30   Median : 3.152
##  Mean   :0.5547   Mean   :6.290   Mean   : 68.38   Mean   : 3.818
##  3rd Qu.:0.6310   3rd Qu.:6.619   3rd Qu.: 94.20   3rd Qu.: 5.215
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127
##       RAD              TAX           PTRATIO            B
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32
##  1st Qu.: 4.000   1st Qu.:280.5   1st Qu.:17.00   1st Qu.:373.81
##  Median : 5.000   Median :334.0   Median :19.00   Median :391.27
##  Mean   : 9.595   Mean   :408.7   Mean   :18.42   Mean   :357.19
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.02
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90
##      LSTAT             MEDV
##  Min.   : 1.920   Min.   : 5.00
##  1st Qu.: 7.065   1st Qu.:17.05
##  Median :11.320   Median :21.20
##  Mean   :12.556   Mean   :22.52
##  3rd Qu.:16.455   3rd Qu.:25.00
##  Max.   :37.970   Max.   :50.00
```

4.1.4 Correlation

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related (www.surveysystem.com).
Below is a reveiw of the correlation between all of the numeric attributes.

```
# Correlation of the attributes
cor(DTset[,1:13])
```

```
##                CRIM          ZN      INDUS         CHAS        NOX
## CRIM     1.00000000 -0.19958871  0.4075617 -0.055071619  0.4099499
## ZN      -0.19958871  1.00000000 -0.5314446 -0.029869089 -0.5202171
## INDUS    0.40756169 -0.53144458  1.0000000  0.065834605  0.7732538
## CHAS    -0.05507162 -0.02986909  0.0658346  1.000000000  0.0933997
## NOX      0.40994989 -0.52021705  0.7732538  0.093399699  1.0000000
## RM      -0.19398049  0.31106841 -0.3826220  0.126767331 -0.2960506
## AGE      0.35240196 -0.58447039  0.6511547  0.073501472  0.7338299
## DIS     -0.37564969  0.67986750 -0.7113431 -0.099046145 -0.7693426
## RAD      0.60834345 -0.32273403  0.6199844 -0.002445292  0.6276047
## TAX      0.57112501 -0.31839453  0.7185102 -0.030644325  0.6757625
## PTRATIO  0.28970459 -0.38877931  0.3781852 -0.122831776  0.1887657
## B       -0.34424053  0.17471792 -0.3643714  0.037822587 -0.3683865
## LSTAT    0.42294190 -0.42188931  0.6135914 -0.084304578  0.5838934
##                RM         AGE         DIS          RAD         TAX
## CRIM    -0.1939805  0.35240196 -0.37564969  0.608343448  0.57112501
## ZN       0.3110684 -0.58447039  0.67986750 -0.322734034 -0.31839453
## INDUS   -0.3826220  0.65115470 -0.71134308  0.619984390  0.71851024
## CHAS     0.1267673  0.07350147 -0.09904614 -0.002445292 -0.03064433
## NOX     -0.2960506  0.73382994 -0.76934256  0.627604694  0.67576249
## RM       1.0000000 -0.22615708  0.20745239 -0.221261630 -0.29526853
## AGE     -0.2261571  1.00000000 -0.74924225  0.468963375  0.50581099
## DIS      0.2074524 -0.74924225  1.00000000 -0.503720701 -0.52641543
## RAD     -0.2212616  0.46896338 -0.50372070  1.000000000  0.92005320
## TAX     -0.2952685  0.50581099 -0.52641543  0.920053202  1.00000000
## PTRATIO -0.3648317  0.27092980 -0.22794289  0.479714803  0.46905607
## B        0.1259942 -0.27418512  0.28427693 -0.423139153 -0.43026638
## LSTAT   -0.6119520  0.60664241 -0.50129941  0.502511782  0.55382142
##            PTRATIO           B       LSTAT
## CRIM     0.2897046 -0.34424053  0.42294190
## ZN      -0.3887793  0.17471792 -0.42188931
## INDUS    0.3781852 -0.36437143  0.61359140
## CHAS    -0.1228318  0.03782259 -0.08430458
## NOX      0.1887657 -0.36838649  0.58389344
## RM      -0.3648317  0.12599418 -0.61195203
## AGE      0.2709298 -0.27418512  0.60664241
## DIS     -0.2279429  0.28427693 -0.50129941
## RAD      0.4797148 -0.42313915  0.50251178
## TAX      0.4690561 -0.43026638  0.55382142
## PTRATIO  1.0000000 -0.16996117  0.40927864
## B       -0.1699612  1.00000000 -0.35088088
## LSTAT    0.4092786 -0.35088088  1.00000000
```

The above results show that a numbers of the atrributes as listed below have a high correlation.

- NOX & INDUS have a correlation of 0.77
- DIS & INDUS have a correlation of 0.71
- AGE & NOX have a correlation of 0.73
- DIS & NOX have a correlation of 0.77
- TAX & RAD have a correlation of 0.92

This high correlation is known as Collinearity. Collinearity in statistics, correlation between predictor variables (or independent variables), such that they express a linear relationship in a regression model. When predictor variables in the same regression model are correlated, they cannot independently predict the value of the dependent variable. In other words, they explain some of the same variance in the dependent variable, which in turn reduces their statistical significance (Felicity, 2019).

It would be better if we were to remove this collinearity to improve regression algorithms results.

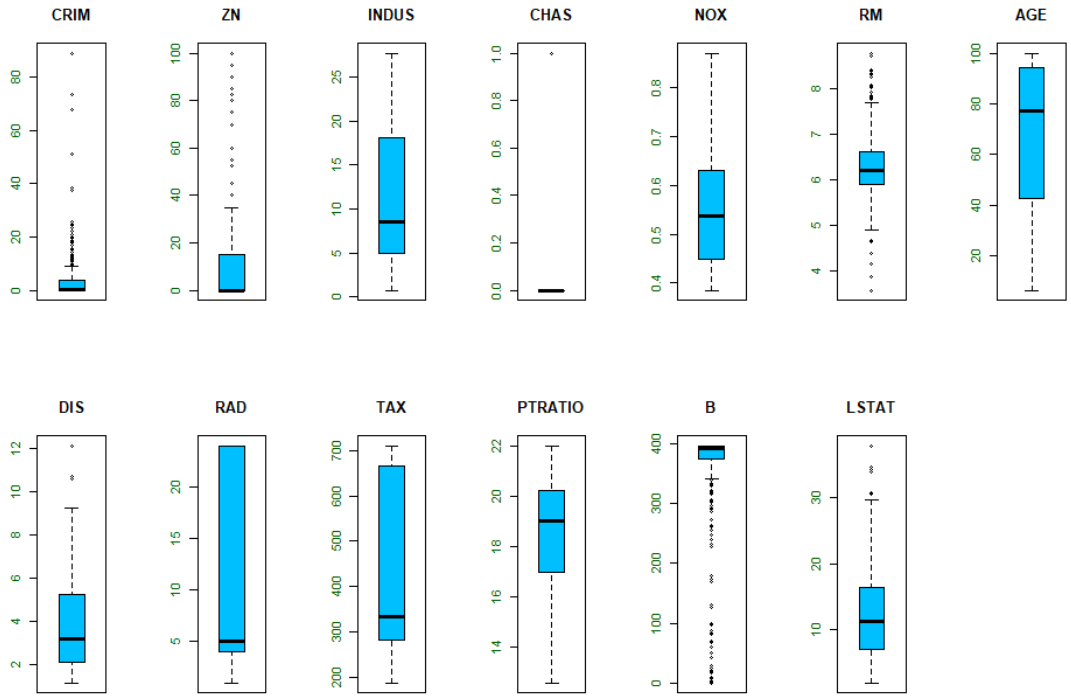
4.2 VISUALISE THE DATA

Data visualization is the graphic representation of data. It involves producing images that communicate relationships among the represented data to viewers of the images. This communication is achieved through the use of a systematic mapping between graphic marks and data values in the creation of the visualization (Wikipedia, 2019)

We can now review and visualise the data.

4.2.1 Box/Whisker Plots

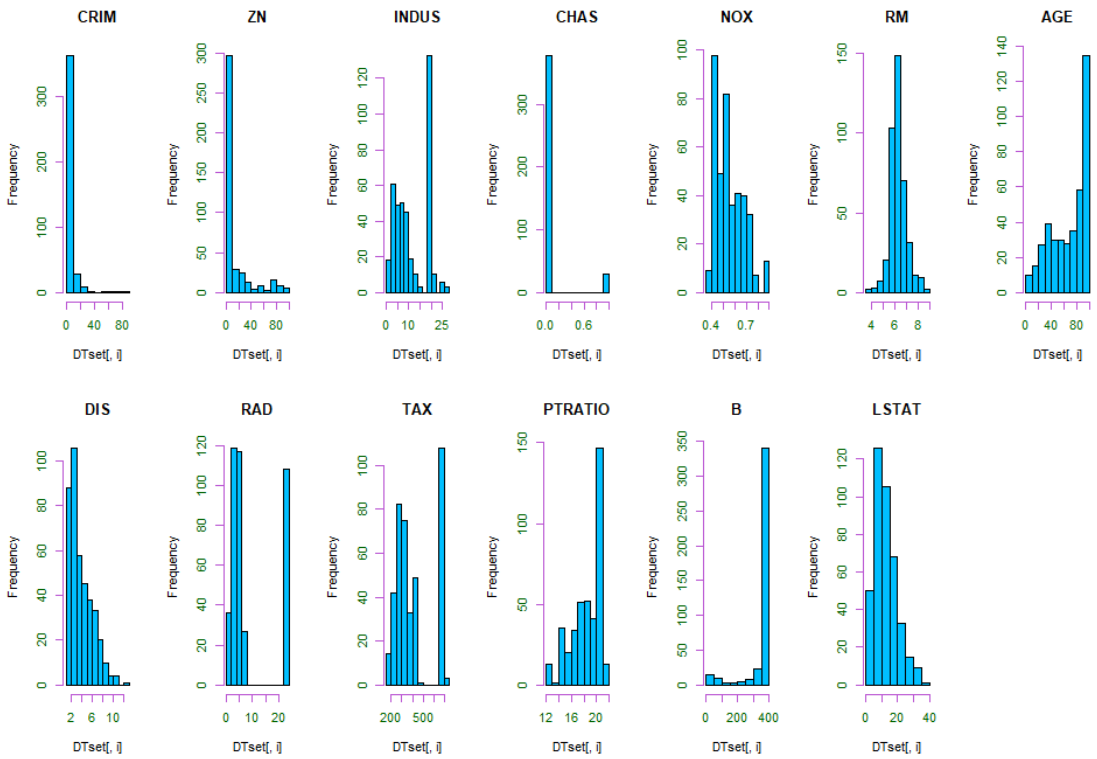
```
# Boxplot visual of the attributes
par(mfrow=c(2,7))
for(i in 1:13){
  boxplot(DTset[,i], main=names(DTset)[i], col="deepskyblue", fg="mediumorchid", col.
axis="darkgreen")
}
```



The above results represents large varainces in the distributions. The data looks like outliers, outside the whisker of the plots.

4.2.2 Histograms

```
# histograms each attribute
par(mfrow=c(2,7))
for(i in 1:13) {
hist(DTset[,i], main=names(DTset)[i], col="deepskyblue", fg="mediumorchid", col.axis=
"darkgreen")
}
```

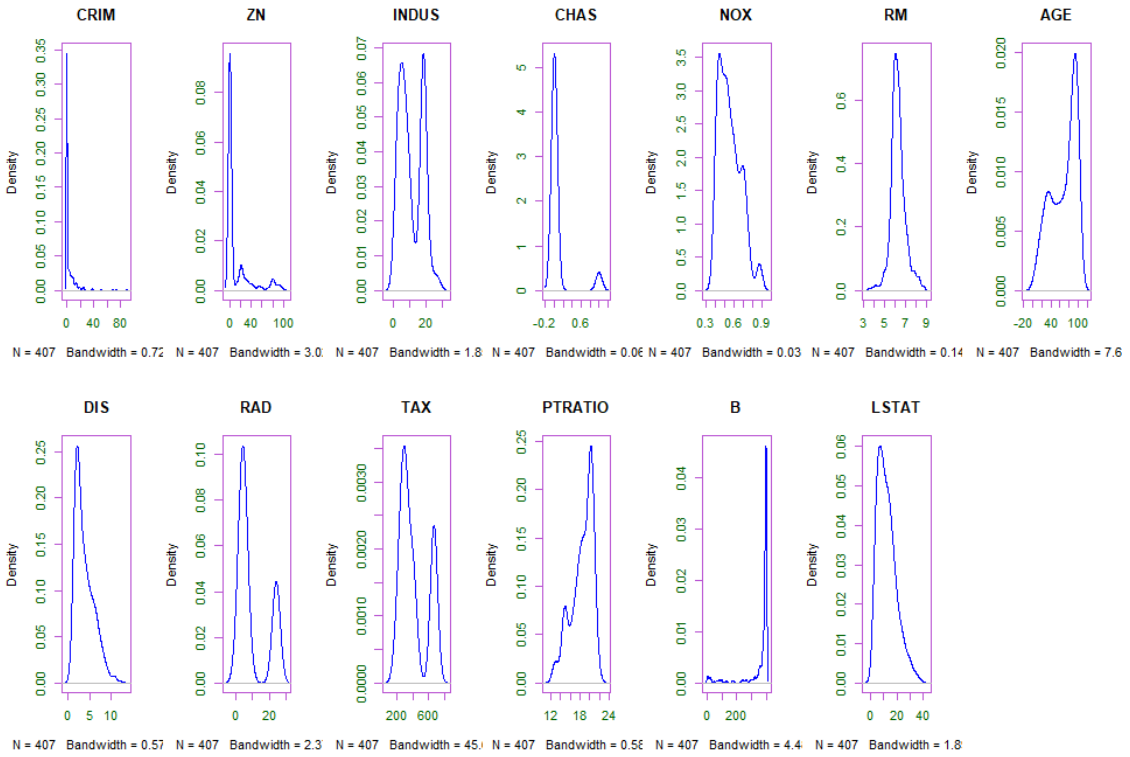


The above results show;

- B, Age, ZN and CRIM has exponential distribution ()
- TAX and RAD has bimodal distribution

4.2.3 Density Plots

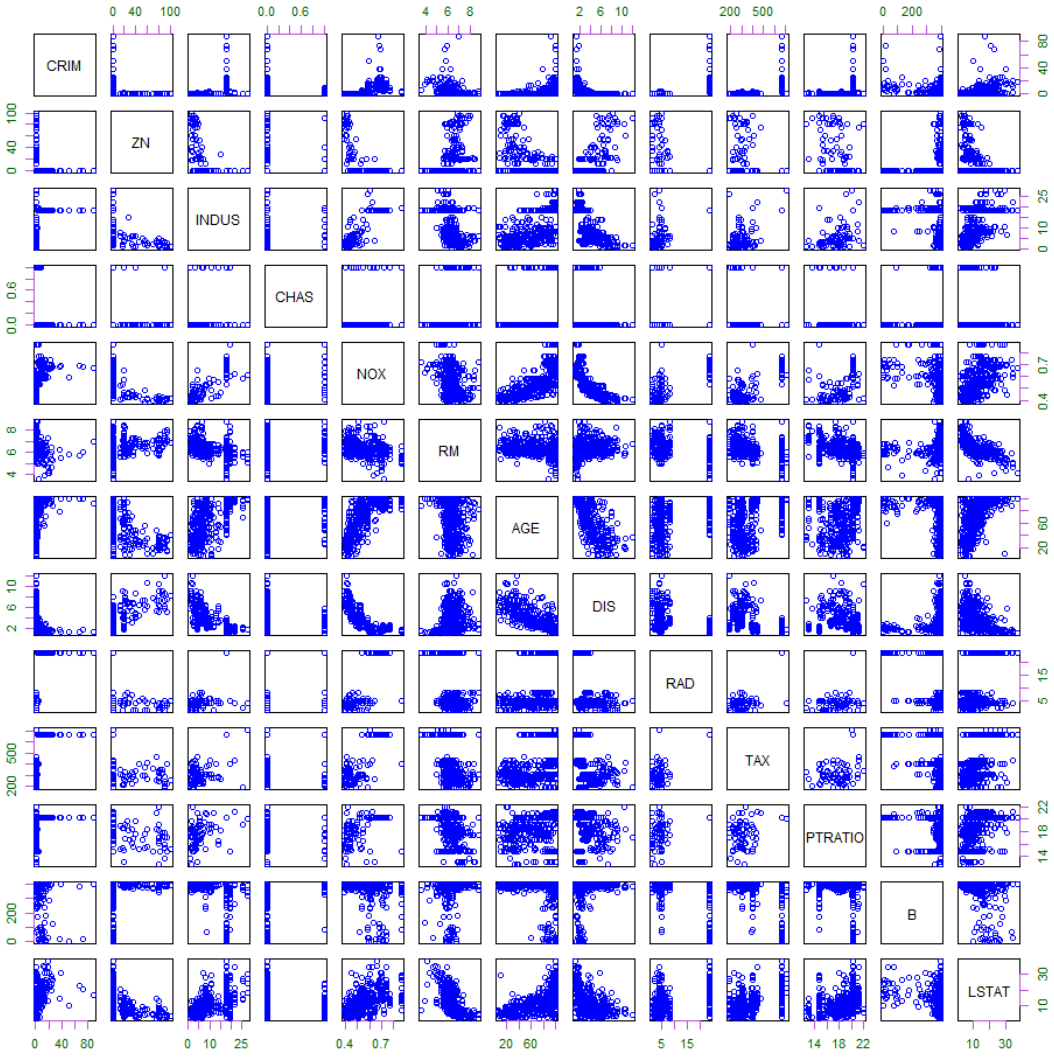
```
# density plot for each attribute
par(mfrow=c(2,7))
for(i in 1:13) {
plot(density(DTset[,i]), main=names(DTset)[i], col="blue", fg="mediumorchid", col.axi
s="darkgreen")
}
```



The above density plots suggest that LSTAT, RM and NOX are like skewed Normal/Gaussian distributions.

4.2.4 Scatter Plots

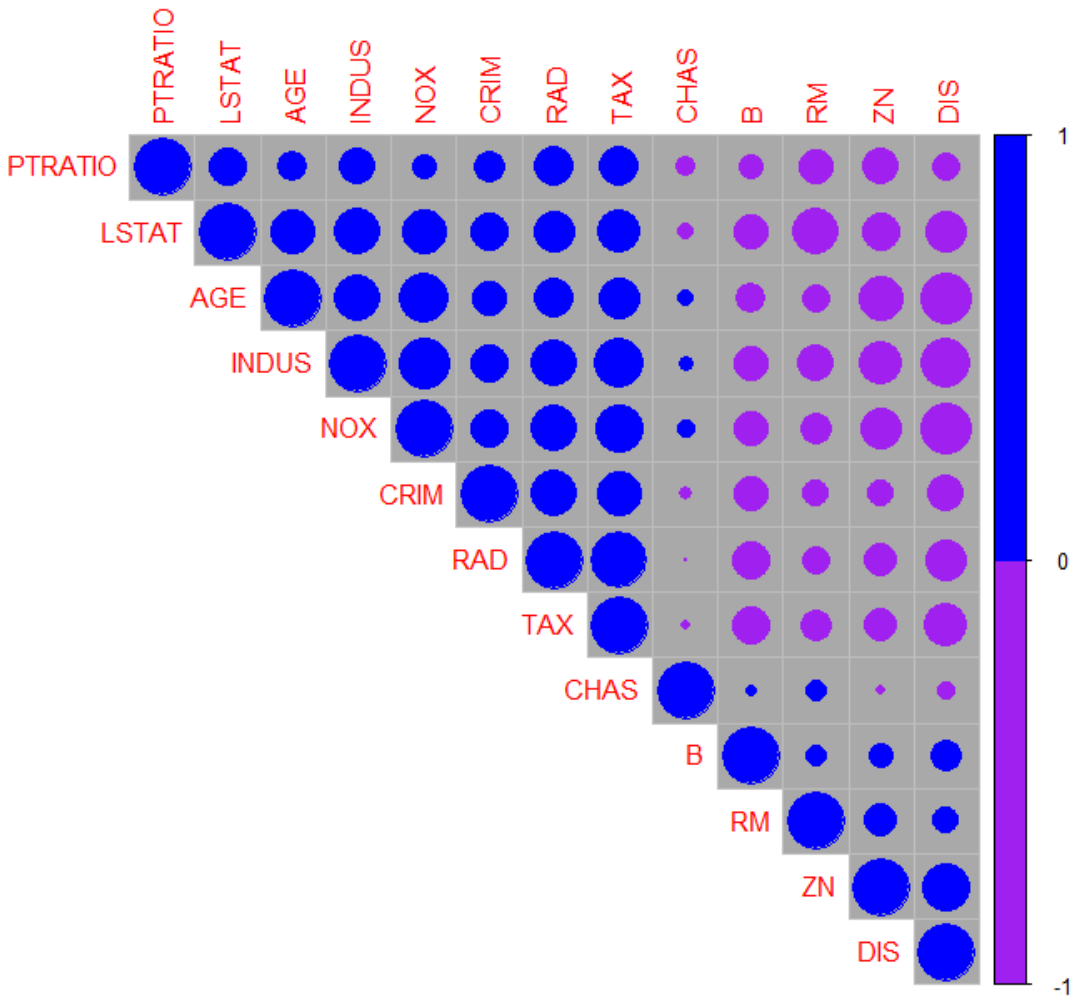
```
pairs(DTset[,1:13], col="blue", fg="mediumorchid", col.axis="darkgreen")
```



We note the structure of the correlated attributes as predictive relationships.

4.2.5 Correlated Plot

```
# correlated plot
correlated<- cor(DTset[,1:13])
corrplot(correlated, type = "upper", order = "hclust", col = c("purple", "blue"), bg
= "darkgrey")
```



As we can see from this plot the large blue circles are positively correlated where-as the larger purple circles are negatively correlated. We need to think of removing these to improve the accuracy of predictive models.

4.3 Observation and next steps

We have reviewed the data and note the detail around the structure of the dataset. In order to improve modelling accuracy we need to modify and transform our data e.g.:-

- Reduce the impact of the differing scales by normalising the dataset
- Reduce the effect of differing distributions via dataset standardisation
- Apply Box Cox transformation to normalise and flatten out some of the distributions to improve accuracy
- Remove highly correlated attributes and feature selection

5 Evaluate Baseline Algorithm

We need to test out a few algorithms to find out which one works well.

Firstly we use 10 fold cross validation with 3 repeats. Our data is a good size set to use for testing. We will use RMSE go get an understanding how wrong our predictions are and R2 will be used to understand how well the model has fit the data.

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

Next we create a baseline of performance and check a number of different algorithms.

Linear Algorithms we use will be;
• LR: Linear A Linear Regression
• GLM: Generalized Linear Regression
• GLMNT: Penalized Linear Regression

Nonlinear Algorithms we use will be;
• CART: Classification and Regression Trees
• SVM: Support Vector Machines
• KNN: k-Nearest Neighbours

```
# LM
set.seed(7)
data.lm <- train(MEDV~., data=DTset, method="lm", metric=metric, preProc=c("center",
"scale"), trControl=trainControl)
# GLM
set.seed(7)
data.glm <- train(MEDV~., data=DTset, method="glm", metric=metric, preProc=c("center"
,
"scale"), trControl=trainControl)
# GLMNET
set.seed(7)
data.glmnet <- train(MEDV~., data=DTset, method="glmnet", metric=metric, preProc=c("c
enter", "scale"), trControl=trainControl)
# SVM
set.seed(7)
data.svm <- train(MEDV~., data=DTset, method="svmRadial", metric=metric,
preProc=c("center", "scale"), trControl=trainControl)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
data.cart <- train(MEDV~., data=DTset, method="rpart", metric=metric, tuneGrid=grid,
preProc=c("center", "scale"), trControl=trainControl)
# KNN
set.seed(7)
data.knn <- train(MEDV~., data=DTset, method="knn", metric=metric, preProc=c("center"
,
"scale"), trControl=trainControl)
```

We now compare our algorithms and summarise the results.

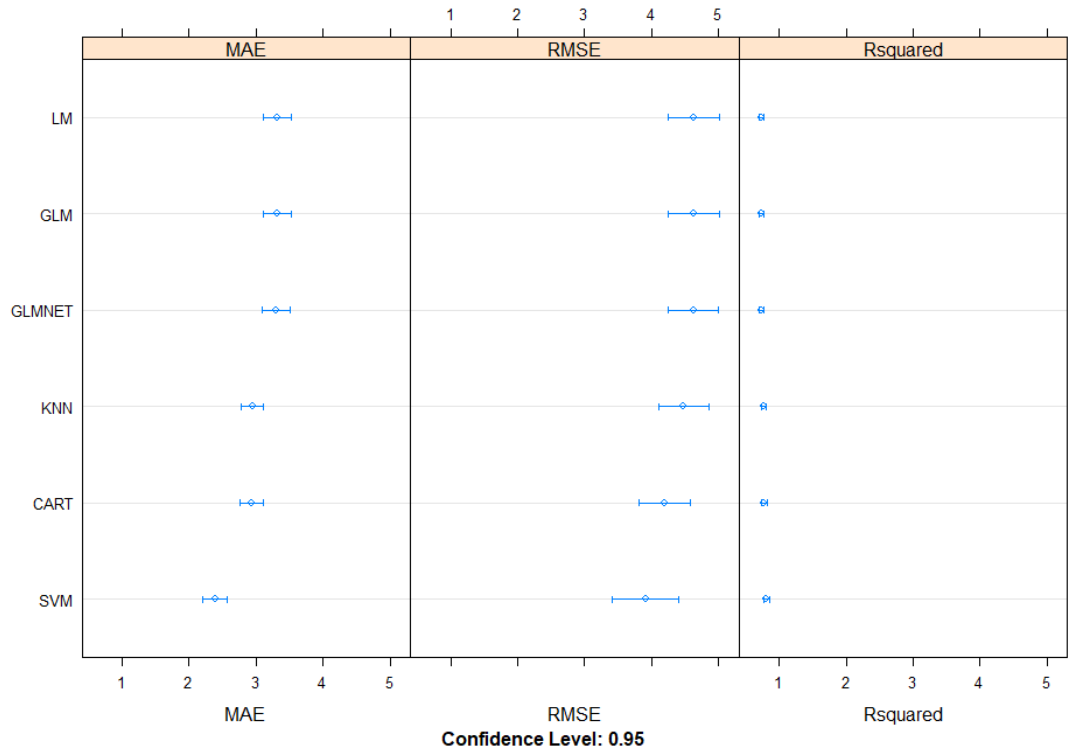
```
# Review our algorithms
baseline.results <- resamples(list(LM=data.lm, GLM=data.glm, GLMNET=data.glmnet, SVM=
data.svm,
CART=data.cart, KNN=data.knn))
summary(baseline.results)

##
## Call:
## summary.resamples(object = baseline.results)
```

```
##
## Models: LM, GLM, GLMNET, SVM, CART, KNN
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.296870 2.897637 3.368197 3.315490 3.701331 4.644634    0
## GLM    2.296870 2.897637 3.368197 3.315490 3.701331 4.644634    0
## GLMNET 2.300353 2.883646 3.336121 3.304888 3.697932 4.625185    0
## SVM    1.422339 1.992016 2.516175 2.387512 2.654770 3.345278    0
## CART   2.216672 2.620729 2.883975 2.933934 3.081861 4.161930    0
## KNN    1.978049 2.685764 2.866806 2.947500 3.237153 3.998333    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.991624 3.869651 4.632032 4.629323 5.317398 6.694651    0
## GLM    2.991624 3.869651 4.632032 4.629323 5.317398 6.694651    0
## GLMNET 2.990832 3.878814 4.615843 4.624746 5.316638 6.692580    0
## SVM    2.049504 2.949764 3.814475 3.908810 4.455599 6.979067    0
## CART   2.766558 3.379400 3.997915 4.199124 4.600681 7.087656    0
## KNN    2.653686 3.741499 4.415526 4.482558 5.064124 6.976913    0
##
## Rsquared
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LM     0.5050763 0.6744319 0.7474419 0.7404282 0.8127596 0.8999946    0
## GLM    0.5050763 0.6744319 0.7474419 0.7404282 0.8127596 0.8999946    0
## GLMNET 0.5033006 0.6730052 0.7474746 0.7410698 0.8155090 0.9039292    0
## SVM    0.5193410 0.7623629 0.8447057 0.8103278 0.8958487 0.9700852    0
## CART   0.5144771 0.7367598 0.8156225 0.7782237 0.8421169 0.8989506    0
## KNN    0.5187652 0.7478699 0.8043069 0.7699680 0.8293089 0.9307667    0
```

The results suggest SVM to have the lowest RMSE followed by CART and KNN. We also note that SVM and other nonlinear algorithms have the best fit for the data in their R2 measures.

Dotplot Visual

```
# Visualise via dotplot
dotplot(baseline.results)
```



6 Feature Selection Algorithms Review

There is a belief that correlated attributes reduce the accuracy of the linear algorithms in the baseline spot-check in the last step. Here we will remove the highly correlated attributes and see what effect this that has on the evaluation metrics.

```
# Exclude correlated attributes

# Look for correlated attributes
set.seed(7)
cutoff <- 0.70
correlations <- cor(DTset[,1:13])
CorrelatedData <- findCorrelation(correlations, cutoff=cutoff)
for(value in CorrelatedData)
  {
print(names(DTset)[value])
}
```

```
## [1] "INDUS"
## [1] "NOX"
## [1] "TAX"
## [1] "DIS"
```

```
# create data exluding correlated features
DTsetFeatures <- DTset[,-CorrelatedData]
dim(DTsetFeatures)
```

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

The above has resulted in excluding these four attritibutes;
1. INDUS
2. NOX
3. TAX
4. DIS

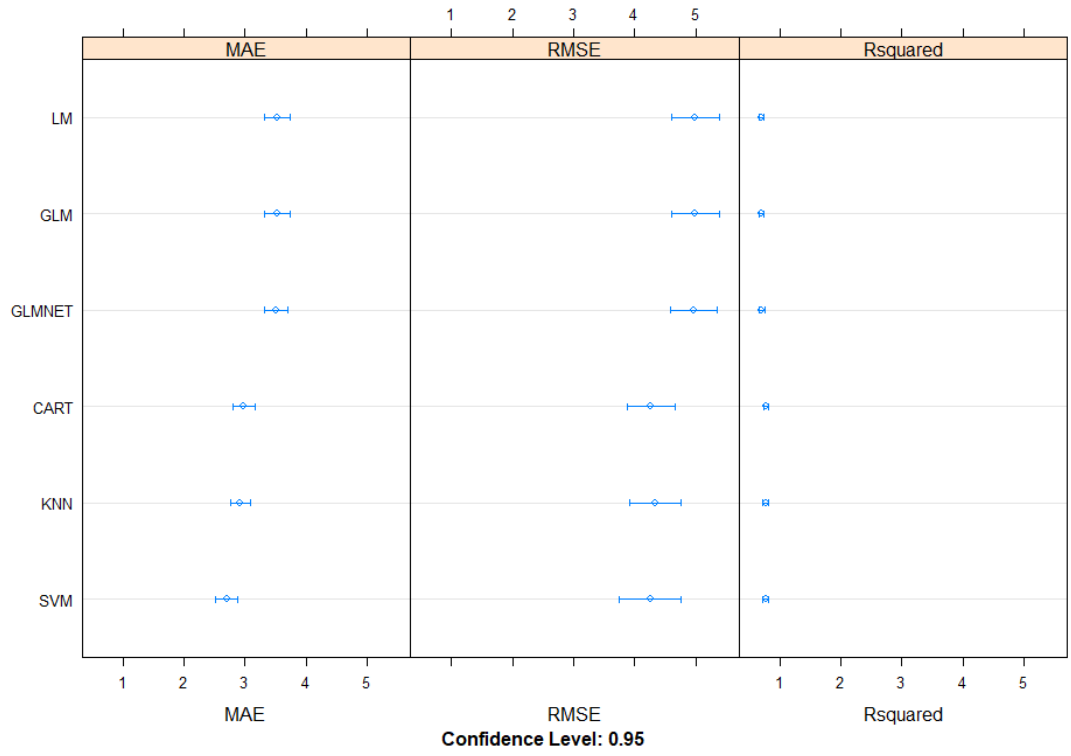
Next we re-run our six algorithms from the baseline.

```
# Algorithms with 10 fold cross validation
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
# lm
set.seed(7)
data.lm <- train(MEDV~., data=DTsetFeatures, method="lm", metric=metric,
preProc=c("center", "scale"), trControl=trainControl)
# GLM
set.seed(7)
data.glm <- train(MEDV~., data=DTsetFeatures, method="glm", metric=metric,
preProc=c("center", "scale"), trControl=trainControl)
# GLMNET
set.seed(7)
data.glmnet <- train(MEDV~., data=DTsetFeatures, method="glmnet", metric=metric,
preProc=c("center", "scale"), trControl=trainControl)
# SVM
set.seed(7)
data.svm <- train(MEDV~., data=DTsetFeatures, method="svmRadial", metric=metric,
preProc=c("center", "scale"), trControl=trainControl)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
data.cart <- train(MEDV~., data=DTsetFeatures, method="rpart", metric=metric,
tuneGrid=grid, preProc=c("center", "scale"), trControl=trainControl)
# KNN
set.seed(7)
data.knn <- train(MEDV~., data=DTsetFeatures, method="knn", metric=metric,
preProc=c("center", "scale"), trControl=trainControl)
# Compare algorithms
feature.results <- resamples(list(LM=data.lm, GLM=data.glm, GLMNET=data.glmnet, SVM=d
ata.svm,
CART=data.cart, KNN=data.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     2.597582 3.198186 3.533505 3.529329 3.810776 4.776058    0
## GLM    2.597582 3.198186 3.533505 3.529329 3.810776 4.776058    0
## GLMNET 2.628837 3.117948 3.542265 3.509583 3.868589 4.690752    0
## SVM    1.870109 2.362839 2.734075 2.702778 2.963509 3.597276    0
## CART   2.081461 2.680047 2.875717 2.979567 3.143345 4.103067    0
## KNN    2.169500 2.603482 2.873902 2.919836 3.205521 3.720976    0
##
```

```
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     3.385745 4.172629 4.913293 4.991283 5.767162 7.146177    0
## GLM    3.385745 4.172629 4.913293 4.991283 5.767162 7.146177    0
## GLMNET 3.382741 4.117021 4.962157 4.971174 5.726988 7.173045    0
## SVM    2.506249 3.229388 4.119990 4.253566 4.983215 7.412917    0
## CART   2.734757 3.571064 3.893022 4.267655 4.762564 7.431916    0
## KNN    2.668598 3.530793 4.046128 4.343244 5.062585 7.029593    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LM     0.4215797 0.6389221 0.6803902 0.7039736 0.8000201 0.8996477    0
## GLM    0.4215797 0.6389221 0.6803902 0.7039736 0.8000201 0.8996477    0
## GLMNET 0.4167207 0.6546938 0.6911644 0.7077446 0.8071089 0.9051059    0
## SVM    0.4694850 0.7139050 0.8125167 0.7732146 0.8608406 0.9460838    0
## CART   0.4852551 0.7379702 0.7967859 0.7710499 0.8474429 0.9176368    0
## KNN    0.4346147 0.6931495 0.7991336 0.7664082 0.8450951 0.9301857    0
```

We note that removal of the correlated attributes are making the RMSE worse for both the linear and non linear algorithms. Removing of correlated attributes improves the accuracy.

```
# Visualise via dotplot
dotplot(feature.results)
```



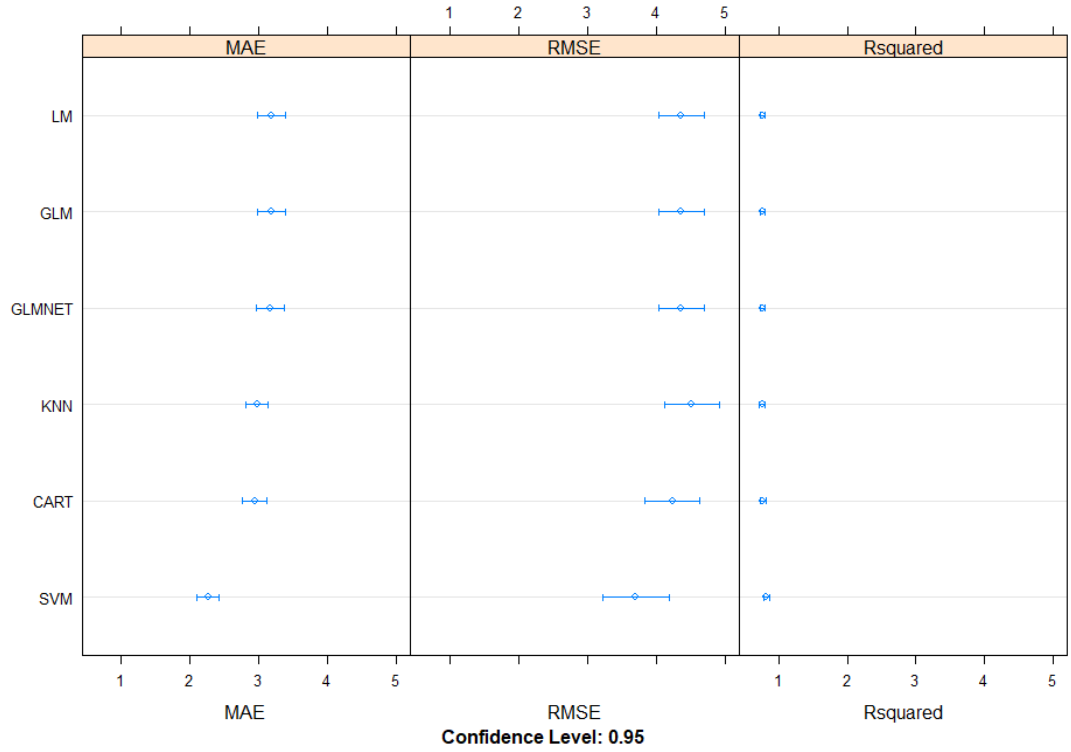
7 Box-Cox Transform Algorithm Review

Our data attributes is skewed with exponential distribution. We will use Box-Cox transformation to rescale the data and review what impact it has on our six algorithms.

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

```
set.seed(7)
data.lm <- train(MEDV~., data=DTset, method="lm", metric=metric, preProc=c("center",
"scale", "BoxCox"), trControl=trainControl)
# GLM
set.seed(7)
data.glm <- train(MEDV~., data=DTset, method="glm", metric=metric, preProc=c("center"
,
"scale", "BoxCox"), trControl=trainControl)
# GLMNET
set.seed(7)
data.glmnet <- train(MEDV~., data=DTset, method="glmnet", metric=metric,
preProc=c("center", "scale", "BoxCox"), trControl=trainControl)
# SVM
set.seed(7)
data.svm <- train(MEDV~., data=DTset, method="svmRadial", metric=metric,
preProc=c("center", "scale", "BoxCox"), trControl=trainControl)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
data.cart <- train(MEDV~., data=DTset, method="rpart", metric=metric, tuneGrid=grid,
preProc=c("center", "scale", "BoxCox"), trControl=trainControl)
# KNN
set.seed(7)
data.knn <- train(MEDV~., data=DTset, method="knn", metric=metric, preProc=c("center"
,
"scale", "BoxCox"), trControl=trainControl)
# Compare algorithms
box.cox.results <- resamples(list(LM=data.lm, GLM=data.glm, GLMNET=data.glmnet, SVM=d
ata.svm,
CART=data.cart, KNN=data.knn))
summary(box.cox.results)

##
## Call:
## summary.resamples(object = box.cox.results)
##
## Models: LM, GLM, GLMNET, SVM, CART, KNN
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.104233 2.797637 3.207088 3.175320 3.454257 4.435940    0
## GLM    2.104233 2.797637 3.207088 3.175320 3.454257 4.435940    0
## GLMNET 2.114563 2.798935 3.209521 3.169094 3.448723 4.432529    0
## SVM    1.295913 1.953074 2.248200 2.259221 2.477442 3.189012    0
## CART   2.216672 2.620729 2.891569 2.939787 3.106475 4.161930    0
## KNN    2.328184 2.664636 2.818222 2.972252 3.271885 3.955278    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.819687 3.811635 4.431132 4.365406 4.995457 6.163792    0
## GLM    2.819687 3.811635 4.431132 4.365406 4.995457 6.163792    0
## GLMNET 2.831297 3.791174 4.421466 4.364087 5.000829 6.184862    0
## SVM    1.804804 2.727022 3.413761 3.702897 4.235294 6.732293    0
## CART   2.766558 3.379400 3.997915 4.234486 4.834258 7.087656    0
## KNN    3.010540 3.725322 4.371531 4.515830 5.022143 7.297372    0
##
## Rsquared
```

```
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LM     0.5598749 0.7120390 0.7753579 0.7661885 0.8254844 0.9096007    0
## GLM    0.5598749 0.7120390 0.7753579 0.7661885 0.8254844 0.9096007    0
## GLMNET 0.5564608 0.7134025 0.7748693 0.7663531 0.8264084 0.9097210    0
## SVM    0.5239104 0.7783632 0.8540991 0.8274159 0.9074418 0.9786169    0
## CART   0.5144771 0.7273216 0.8156225 0.7743938 0.8430400 0.8989506    0
## KNN    0.4922397 0.7231826 0.7919972 0.7622184 0.8419261 0.9365145    0

# Visualise via dotplot
dotplot(box.cox.results)
```



Now we note decreased RMSE and increased R2 on all except the CART algorithms.

8 TUNING TO IMPROVE ACCURACY

To improve the accuracy of the algorithms we can tune their parameters. Here, we will tune SVM parameters with RBF (Radial Basis Function).

Let us see the estimated accuracy of our model.

```
# Print DataSVM fiited model
print(data.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: 365, 366, 366, 367, 366, 366, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
```

```
##   0.25  4.540024  0.7717394  2.731571
##   0.50  4.073572  0.8016482  2.459453
##   1.00  3.702897  0.8274159  2.259221
##
## Tuning parameter 'sigma' was held constant at a value of 0.1160954
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1160954 and C = 1.
```

Next steps;
• We build a grid search around C value of 1
• We may note a reduction in the RMSE with increase in C. We shall try C values of 1 to 10
• CARET package allows the sigma parameter to be tuned for smoothing. Ideally values initiating 0.1 are good.

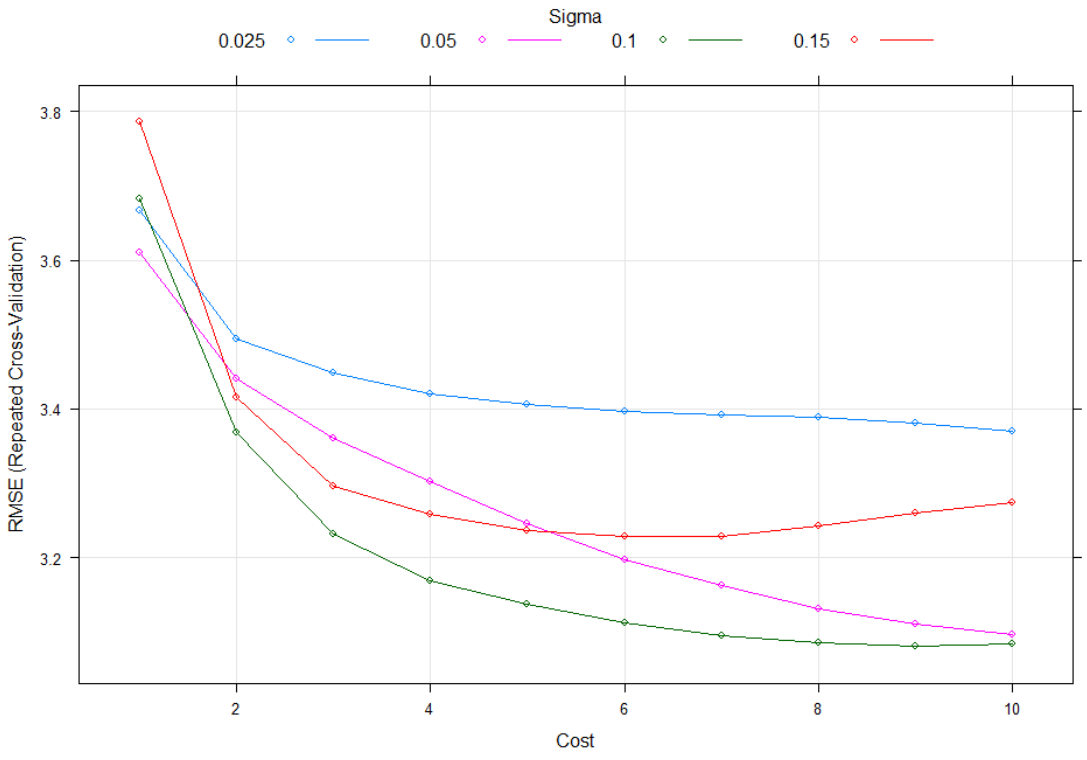
```
# Tunning SVM and C
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
set.seed(7)
grid <- expand.grid(.sigma=c(0.025, 0.05, 0.1, 0.15), .C=seq(1, 10, by=1))
data.svm <- train(MEDV~., data=DTset, method="svmRadial", metric=metric, tuneGrid=gri
d,
preProc=c("BoxCox"), trControl=trainControl)
print(data.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: 365, 366, 366, 367, 366, 366, ...
## Resampling results across tuning parameters:
##
##   sigma  C   RMSE      Rsquared   MAE
##   0.025   1  3.666043  0.8304251  2.341464
##   0.025   2  3.493628  0.8402429  2.210608
##   0.025   3  3.448027  0.8424578  2.166475
##   0.025   4  3.421134  0.8441250  2.144822
##   0.025   5  3.406023  0.8452104  2.126361
##   0.025   6  3.396596  0.8459419  2.118039
##   0.025   7  3.392730  0.8458421  2.112808
##   0.025   8  3.389535  0.8456315  2.109806
##   0.025   9  3.381037  0.8459219  2.107856
##   0.025  10  3.369434  0.8466299  2.103481
##   0.050   1  3.610080  0.8333388  2.253360
##   0.050   2  3.441528  0.8431029  2.172318
##   0.050   3  3.360002  0.8478343  2.111491
##   0.050   4  3.302564  0.8519628  2.081684
##   0.050   5  3.245578  0.8563020  2.054521
##   0.050   6  3.197392  0.8597917  2.032993
##   0.050   7  3.162971  0.8623836  2.024649
##   0.050   8  3.132003  0.8645481  2.016135
##   0.050   9  3.111908  0.8658503  2.009976
##   0.050  10  3.097578  0.8667604  2.006334
##   0.100   1  3.682911  0.8286388  2.256098
##   0.100   2  3.369351  0.8482277  2.120290
##   0.100   3  3.231527  0.8577559  2.059342
```

```
##   0.100   4  3.168919  0.8623427  2.043215
##   0.100   5  3.137567  0.8645176  2.038875
##   0.100   6  3.112662  0.8663073  2.037551
##   0.100   7  3.094980  0.8675240  2.038389
##   0.100   8  3.086051  0.8679036  2.039479
##   0.100   9  3.080897  0.8682378  2.040387
##   0.100  10  3.084181  0.8680824  2.045987
##   0.150   1  3.785244  0.8219277  2.301245
##   0.150   2  3.415474  0.8464298  2.141562
##   0.150   3  3.296697  0.8543834  2.090340
##   0.150   4  3.258625  0.8567102  2.088175
##   0.150   5  3.236921  0.8575635  2.092287
##   0.150   6  3.228499  0.8576468  2.102106
##   0.150   7  3.228241  0.8573974  2.115081
##   0.150   8  3.242664  0.8563191  2.133669
##   0.150   9  3.260328  0.8550405  2.153824
##   0.150  10  3.273906  0.8540664  2.170337
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1 and C = 9.
```

In the results we note that sigma values flatten out with larger C cost. The final values used for the model were sigma = 0.1 and C = 9.

```
# Plot tunning results
plot(data.svm)
```



9 ENSEMBLE METHODS ALOGRITHMS

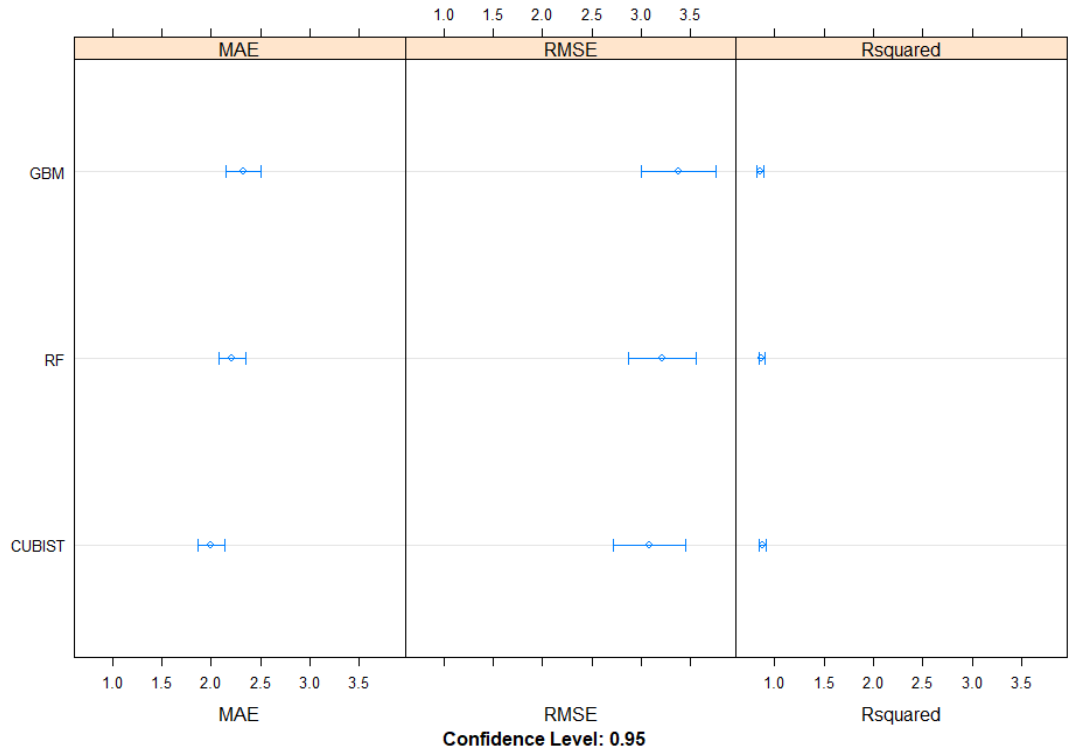
In this section we will attempt to further reduce RMSE by applying ensemble methods like boosting and bagging techniques for decision trees.

Here we apply;
• RF; Random Forest, bagging
• CUBIST boosting
• GBM; Gradient Boosting Machines

```
# Ensembles
seed <- 7
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
# RF
set.seed(seed)
ensemble.rf <- train(MEDV~., data=DTset, method="rf", metric=metric, preProc=c("BoxCo
x"),
trControl=trainControl)
# CUBIST
set.seed(seed)
ensemble.cubist <- train(MEDV~., data=DTset, method="cubist", metric=metric,
preProc=c("BoxCox"), trControl=trainControl)
# GBM
set.seed(seed)
ensemble.gbm <- train(MEDV~., data=DTset, method="gbm", metric=metric, preProc=c("Box
Cox"),
trControl=trainControl, verbose=FALSE)

# Evaluate resultsalgorithms
ensembles.output <- resamples(list(RF=ensemble.rf, CUBIST=ensemble.cubist, GBM=ensemb
le.gbm))
summary(ensembles.output)

##
## Call:
## summary.resamples(object = ensembles.output)
##
## Models: RF, CUBIST, GBM
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## RF     1.636635 1.977145 2.200994 2.210221 2.302417 3.220658    0
## CUBIST 1.311433 1.754070 1.952186 2.000247 2.173831 2.891565    0
## GBM    1.646824 1.979192 2.265152 2.327776 2.545692 3.752348    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## RF     2.112991 2.575385 3.076901 3.218102 3.695468 6.521859    0
## CUBIST 1.786686 2.379441 2.738151 3.087677 3.779192 5.788691    0
## GBM    1.917678 2.544770 3.310917 3.379109 3.665515 6.851109    0
##
## Rsquared
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF     0.5972355 0.8517322 0.9004050 0.8685981 0.9188907 0.9724223    0
## CUBIST 0.6805940 0.8176997 0.9092059 0.8754041 0.9318374 0.9699312    0
## GBM    0.5581867 0.8154935 0.8886156 0.8508565 0.9120289 0.9643755    0
```

The results show us that Cubist resulted in the lowest RMSE than all the others.

```
# Plot tunning results
dotplot(ensembles.output)
```



Next, we into tuning Cubist further by using two of the Caret parameters;

• Committees: The tree-based Cubist model can be easily used to develop an ensemble classifier with a scheme called "committees". The concept of "committees" is similar to the one of "boosting" by developing a series of trees sequentially with adjusted weights. However, the final prediction is the simple average of predictions from all "committee" members, an idea more close to "bagging" (Statcompute, 2015)

• Neighbors; numbers of instances used for prediction

Detailed review of Cubist

```
# Detailed review of Cubist
print(ensemble.cubist)

## Cubist
##
## 407 samples
##  13 predictor
##
## Pre-processing: Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 365, 366, 366, 367, 366, 366, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE      Rsquared   MAE
##    1          0          3.935283  0.8050980  2.501518
##    1          5          3.663278  0.8278677  2.239888
##    1          9          3.685088  0.8254836  2.257340
##   10          0          3.449718  0.8480935  2.288926
##   10          5          3.191849  0.8675510  2.044831
##   10          9          3.229558  0.8643281  2.074724
##   20          0          3.339729  0.8576020  2.247981
##   20          5          3.087677  0.8754041  2.000247
##   20          9          3.122622  0.8723800  2.031336
##
```

```
## 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.
```

As we note from the above summary, the final values used for the model were committees = 20 and neighbors = 5.

Next, we apply grid search to tune our values using all committees, 15 to 25. We also review for a neighbors over and under 5.

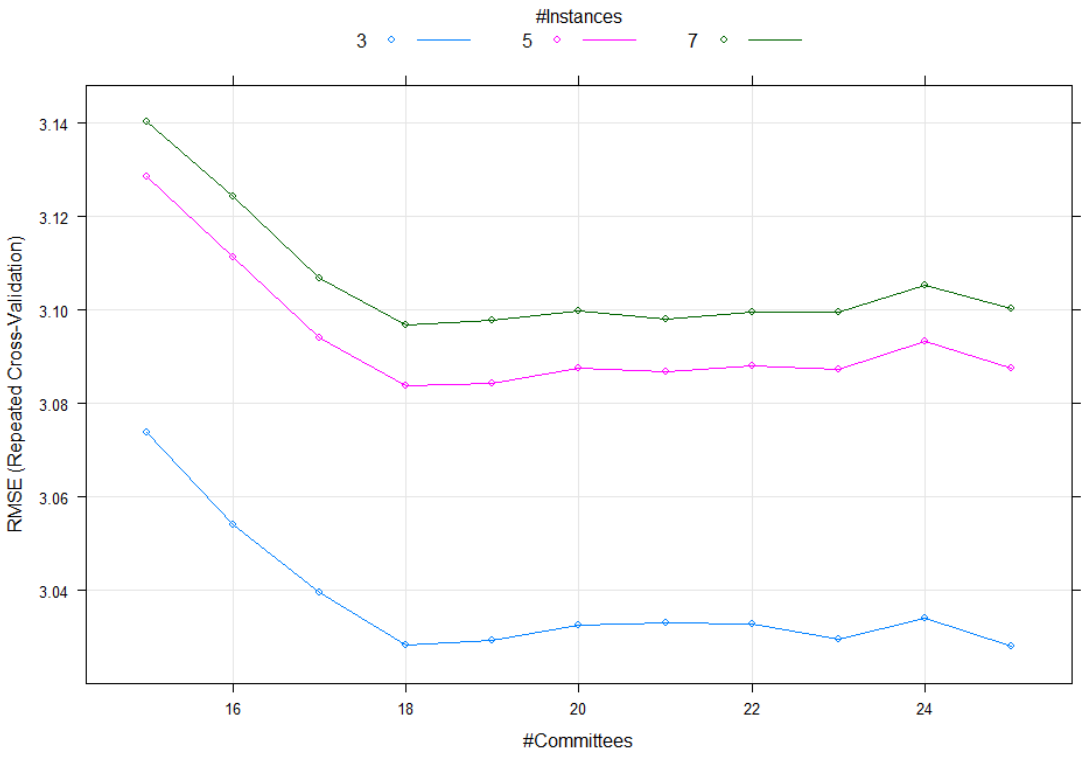
```
# Tunning Cubist
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
set.seed(7)
grid <- expand.grid(.committees=seq(15, 25, by=1), .neighbors=c(3, 5, 7))
cubist.tuned <- train(MEDV~., data=DTset, method="cubist", metric=metric,
preProc=c("BoxCox"), tuneGrid=grid, trControl=trainControl)
print(cubist.tuned)
```

```
## Cubist
##
## 407 samples
##  13 predictor
##
## Pre-processing: Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 365, 366, 366, 367, 366, 366, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE      Rsquared   MAE
##   15          3          3.073937  0.8767747  1.995981
##   15          5          3.128523  0.8727199  2.016761
##   15          7          3.140179  0.8711152  2.029532
##   16          3          3.054057  0.8778157  1.988389
##   16          5          3.111381  0.8736537  2.010829
##   16          7          3.124320  0.8719441  2.024607
##   17          3          3.039552  0.8788573  1.979358
##   17          5          3.094127  0.8748690  2.000618
##   17          7          3.106775  0.8731719  2.013405
##   18          3          3.028333  0.8797190  1.973590
##   18          5          3.083740  0.8757237  1.997690
##   18          7          3.096770  0.8740027  2.012345
##   19          3          3.029223  0.8797359  1.971444
##   19          5          3.084257  0.8757493  1.994837
##   19          7          3.097699  0.8740371  2.008627
##   20          3          3.032489  0.8793502  1.977003
##   20          5          3.087677  0.8754041  2.000247
##   20          7          3.099866  0.8737809  2.014252
##   21          3          3.032965  0.8794883  1.978889
##   21          5          3.086705  0.8755892  2.002846
##   21          7          3.098157  0.8740286  2.016069
##   22          3          3.032896  0.8793041  1.980806
##   22          5          3.087947  0.8754550  2.003035
##   22          7          3.099654  0.8739021  2.016758
##   23          3          3.029535  0.8797109  1.980914
##   23          5          3.087364  0.8756108  2.006294
##   23          7          3.099492  0.8740565  2.020778
##   24          3          3.034052  0.8792030  1.982144
##   24          5          3.093291  0.8750003  2.006240
##   24          7          3.105403  0.8734220  2.021364
```

```
##   25          3          3.027991  0.8800070  1.982355
##   25          5          3.087580  0.8757971  2.006807
##   25          7          3.100416  0.8741655  2.021189
##
## 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.
```

Our RMSE improved to 3.027991 with committees = 25 and neighbors = 3.

```
# Plot tunned Cubist
plot(cubist.tuned)
```



10 Optimal Algorithm

From our modelling and its outcomes, Cubist seems to the accurate one.

Next, we generate an independent Cubist model from the above parameters with the complete dataset using Box-Cox transformation.

```
# Data preparation
set.seed(7)
x <- DTset[,1:13]
y <- DTset[,14]
prep.Parms <- preProcess(x, method=c("BoxCox"))
transformX <- predict(prep.Parms, x)
# training Cubist Model
cubist.model <- cubist(x=transformX, y=y, committees=18)
```

The view the full summary we can insert this code into our RMD chunk;
summary(cubist.model)

```
# Transforming validation dataset
set.seed(7)
```

```
validation.X <- validateSet[,1:13]
transform.validation.X <- predict(prep.Parms, validation.X)
validation.Y <- validateSet[,14]
# Predictions based on Cubist model with the validation data
predict.output <- predict(cubist.model, newdata=transform.validation.X, neighbors=3)
# Final RMSE
Final.rmse <- RMSE(predict.output, validation.Y)
r_2 <- R2(predict.output, validation.Y)
print(Final.rmse)
```

```
## [1] 3.237403
```

We note that the RMSE output from our Cubist model is 3.237403 which is a little low but quite similar to the expected RMSE of 3.027991.

11 Conclusion

We have applied a regression predictive machine learning model in this project. We used the Boston housing data where we first analysed the data where we found skewed distributions and correlations. Next, we evaluated a number of algorithms, removed correlated attributes, transformed & tuned the data. Finally we applied ensemble methods of bagging and boosting on the Cubist.

12 References

12.1 Articles

• Irizzary,R., "Introduction to Data Science,github page,*https://rafalab.github.io/dsbook/*", (2018)
• Felicity Boyd Enders, "Collinearity, Statistics"- *https://www.britannica.com/topic/collinearity-statistics*
• Data visualization, Wikipedia - *https://en.wikipedia.org/wiki/Data_visualization*, 2019
• Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The New S Language. Wadsworth & Brooks/Cole
• Vishal R, "Feature selection — Correlation and P-value", 2018

12.2 Web Links

• *https://holtzy.github.io/Pimp-my-rmd/*
• *https://www.surveysystem.com/correlation.htm*
• *http://www.stat.tamu.edu/~hart/652/collinear.pdf*
• *https://www.britannica.com/topic/collinearity-statistics*
• *https://www.statisticshowto.datasciencecentral.com/box-cox-transformation/*
• *https://towardsdatascience.com/feature-selection-correlation-and-p-value-da8921bfb3cf*
• *https://statcompute.wordpress.com/2015/03/21/ensemble-learning-with-cubist-model/*
• *https://static1.squarespace.com/static/51156277e4b0b8b2ffe11c00/t/56e3056a3c44d8779a61988a/1457718645593/cubist_BRUG.pdf*