Capstone Project

Astrid Melhado Dyer

9/18/2019

Contents

For this submission, I will explore the BostonHouse2 data set. Fully interested in developing predictive models for sustainable low-income housing projects, the current data set is fit to the purpose. The goal as prescribed by the assignment prompt is to create a machine learning model to showcase the data scientist skills gained through the learning path.

0.1 Problem Definition

A regression task meant to create a machine learning model to predict the median value of owner-occupied homes in USD 1000's (cmedv-target variable) based on 18 features pertaining to the BostonHousing2 data set

0.2 General Considerations

Due to the pedagogic nature of the assignment, selected pieces of code are displayed when deemed appropriate. On the other hand, to not interrupt the report flow, the eval parameter of the r chunks corresponding to the final model have been set to FALSE as it takes more than 15 pages to be reproduced. However, if suffises in the Rmd to set the eval parameter to TRUE if the entire output is desired.

0.3 The Data

The BostonHousing2 data set is a corrected version of its predecessor: BostonHousing data set by Harrison and Rubinfeld (1979). Originally, it contained housing data for 506 census tracts of Boston from the 1970 census. 5 variables have been added to the corrected version for a total of 506 observations and 19 features.

0.3.1 Data Loading and R packages.

The data corresponds to a curated list of data sets cleaned and ready for machine learning analysis to be found in the mlbench package. Original data set has been taken from the UCI Repository of Machine Learning Databases [1]and the corrected version from Statlib[2].

0.3.2 Create Data Partition

As expected, data is being partitioned in a set of data destined to train the model (train set) and in a set of data(test set) that we hold on until the end of the project to confirm the accuracy of the model on unseen data.

```r
set.seed(13) # for reproducibility's purpose
train_index <- createDataPartition(BostonHousing2$cmedv, p=0.80, list = FALSE)
train_set <- BostonHousing2[train_index,]
test_set <- BostonHousing2[-train_index,]
```

0.4 Descriptive Statistcs

The objective of this segment is to create meaningful summaries about the sample that may form the basis of an extensive statistical analysis in order to craft the best fit.

0.4.1 Quantitative Summary

Looking at the data types of the following attributes, we found two of the features being factors:

```
     town     tract       lon       lat      medv     cmedv      crim 
 "factor" "integer" "numeric" "numeric" "numeric" "numeric" "numeric" 
       zn     indus      chas       nox        rm       age       dis 
"numeric" "numeric"  "factor" "numeric" "numeric" "numeric" "numeric" 
      rad       tax   ptratio         b     lstat 
"integer" "integer" "numeric" "numeric" "numeric" 
```

The first 6 rows of our data set.

cmedv	town	tract	lon	lat	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat
24.0	54	2011	-70.9550	42.2550	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
21.6	77	2021	-70.9500	42.2875	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
34.7	77	2022	-70.9360	42.2830	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
33.4	46	2031	-70.9280	42.2930	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
36.2	46	2032	-70.9220	42.2980	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
28.7	46	2033	-70.9165	42.3040	0.02985	0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21

Measures to be taken:

- All features must be numeric. Transform **town** and **chas** variables
- Remove **medv** column. The later has been the target variable in the original data set. Since we are working on the **BostonHousing2** data set, it has been substituted by **cmedv**.

[1] http://www.ics.uci.edu/~mlearn/MLRepository.html
[2] http://lib.stat.cmu.edu/datasets/

- We order the columns placing the target variable **cmedv** first for for manageability's purposes

Confirming the data dimension is just the beginning.

```
[1] 407  18
```

Now, it is advisable to take a peek of the several distributions.

vars	n	mean	sd	median	trimmed	mad	min	max	range	se	SKEW	KURTOSIS
1	407	40.5012285	26.4024251	35.00000	39.1406728	31.1346000	1.00000	92.0000	91.00000	1.3087196	0.369785329291809	-1.13841857338827
2	407	2717.6560197	1390.1861837	3393.00000	2790.2782875	1183.1148000	1.00000	5082.0000	5081.00000	68.9089708	-0.427482622800659	-1.17342777425525
3	407	-71.0547806	0.0765290	-71.05170	-71.0528807	0.0575249	-71.28950	-70.8100	0.47950	0.0037934	-0.211262317729226	1.1258819709641
4	407	42.2154246	0.0622082	42.21830	42.2163596	0.0566353	42.03000	42.3740	0.34400	0.0030835	-0.137643777335944	-0.0232559635915601
5	407	3.4614171	8.4932535	0.26169	1.6154726	0.3360609	0.00632	88.9762	88.96988	0.4209949	5.72172606887442	44.1388526770897
6	407	11.7579853	23.9276882	0.00000	5.3379205	0.0000000	0.00000	95.0000	95.00000	1.1860515	2.11515673371002	3.36960287331553
7	407	11.0716953	6.8952198	8.56000	10.8550153	7.9467360	0.46000	27.7400	27.28000	0.3417834	0.305838605610626	-1.22923200550695
8	407	0.0663391	0.2491802	0.00000	0.0000000	0.0000000	0.00000	1.0000	1.00000	0.0123514	3.47214986173814	10.0806108437299
9	407	0.5542848	0.1171595	0.53800	0.5443049	0.1319514	0.38500	0.8710	0.48600	0.0058074	0.735843832115009	-0.13655064372043
10	407	6.2918403	0.6874747	6.21100	6.2581865	0.4996362	3.56100	8.7800	5.21900	0.0340769	0.373806025259542	2.14738872471193
11	407	68.5076167	27.9859345	77.70000	71.0275229	29.0589600	2.90000	100.0000	97.10000	1.3872113	-0.575406009401236	-1.00468762099762
12	407	3.8290256	2.1257845	3.27970	3.5682651	1.9270835	1.12960	12.1265	10.99690	0.1053712	1.03045016055932	0.585277096241814
13	407	9.3808354	8.6923287	5.00000	8.5443425	2.9652000	1.00000	24.0000	23.00000	0.4308627	1.02712528886727	-0.818492329603351
14	407	408.2309582	167.6884888	330.00000	399.7859327	108.2298000	187.00000	711.0000	524.00000	8.3120098	0.679024467374782	-1.12698635730105
15	407	18.5223587	2.1549499	19.10000	18.7376147	1.6308600	12.60000	22.0000	9.40000	0.1068169	-0.839032294844186	-0.210207760119886
16	407	358.0159951	90.2594500	391.83000	384.0600000	7.5167820	0.32000	396.9000	396.58000	4.4739948	-2.93295517580642	7.48640733759178
17	407	12.3851843	6.8417960	11.25000	11.7153517	6.9237420	1.73000	37.9700	36.24000	0.3391352	0.827949222484476	0.292250697867562

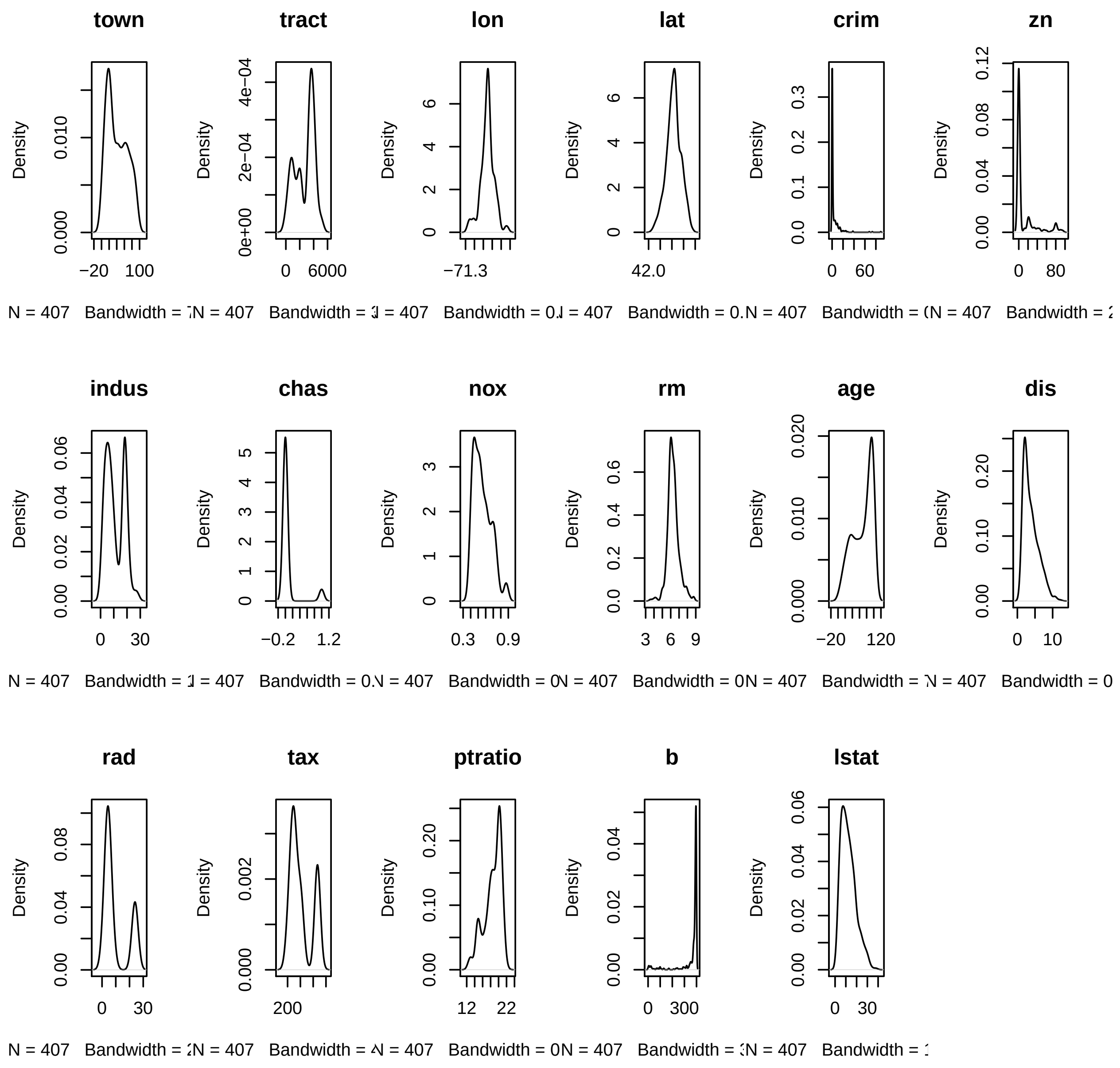
Let us recapitulate our findings, so far:

- Attributes' scales are all over the place
- Some features exhibit moderate to high skewness; signaled in red.
- Some features exhibit noticeable kurtosis, mainly leptokurtic distributions.

0.4.2 Visualizations

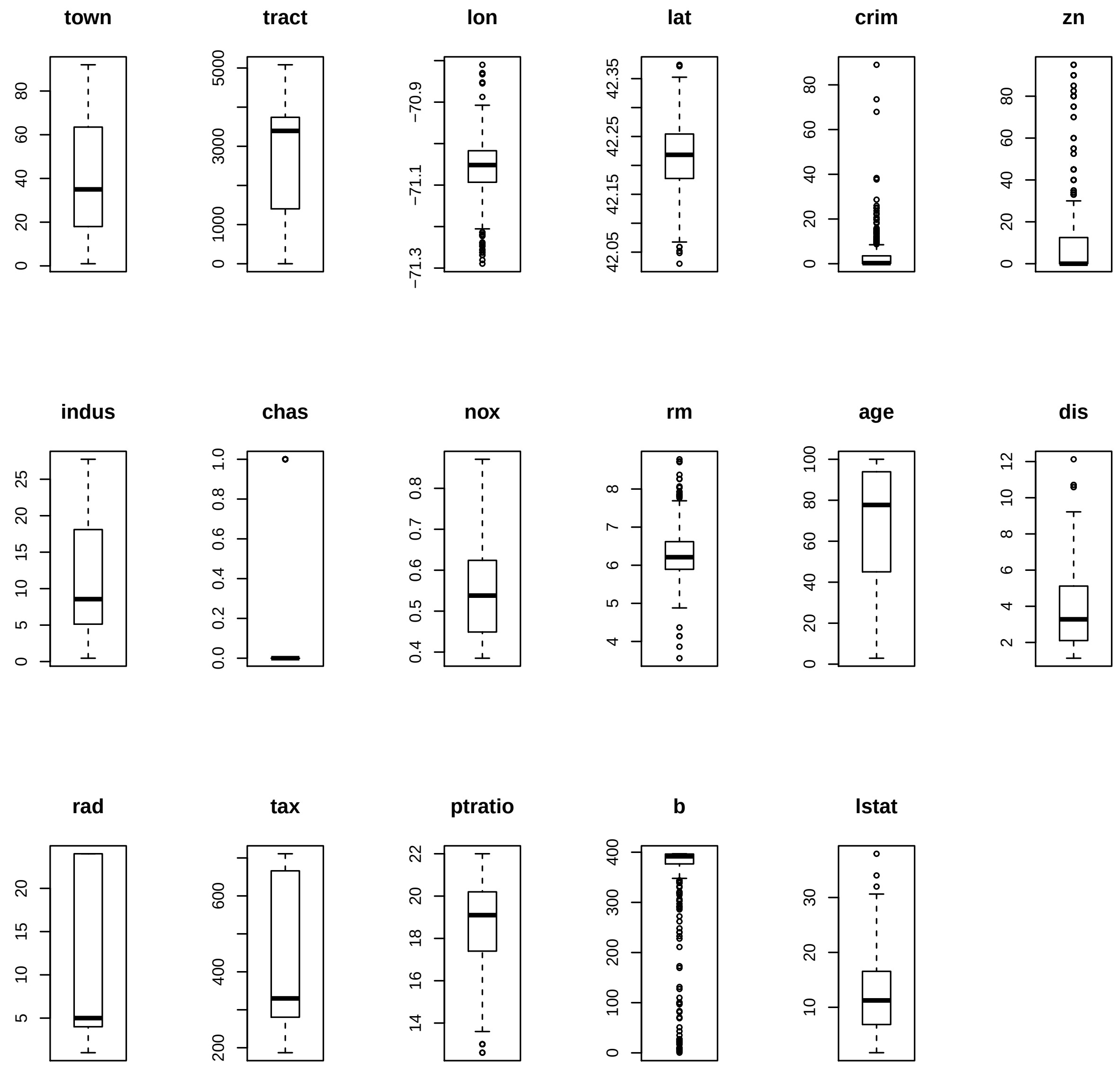
The following display evidences the skewness already seen in some of the distributions It looks like **rad** and **tax** are clearly bimodal, and some columns might show exponential distributions, as well.

0.4.2.1 Density Plots



To confirm our suspicions, let us glimpse for the outliers beyond the whiskers of the enclosed boxplots:

0.4.2.2 Boxplots



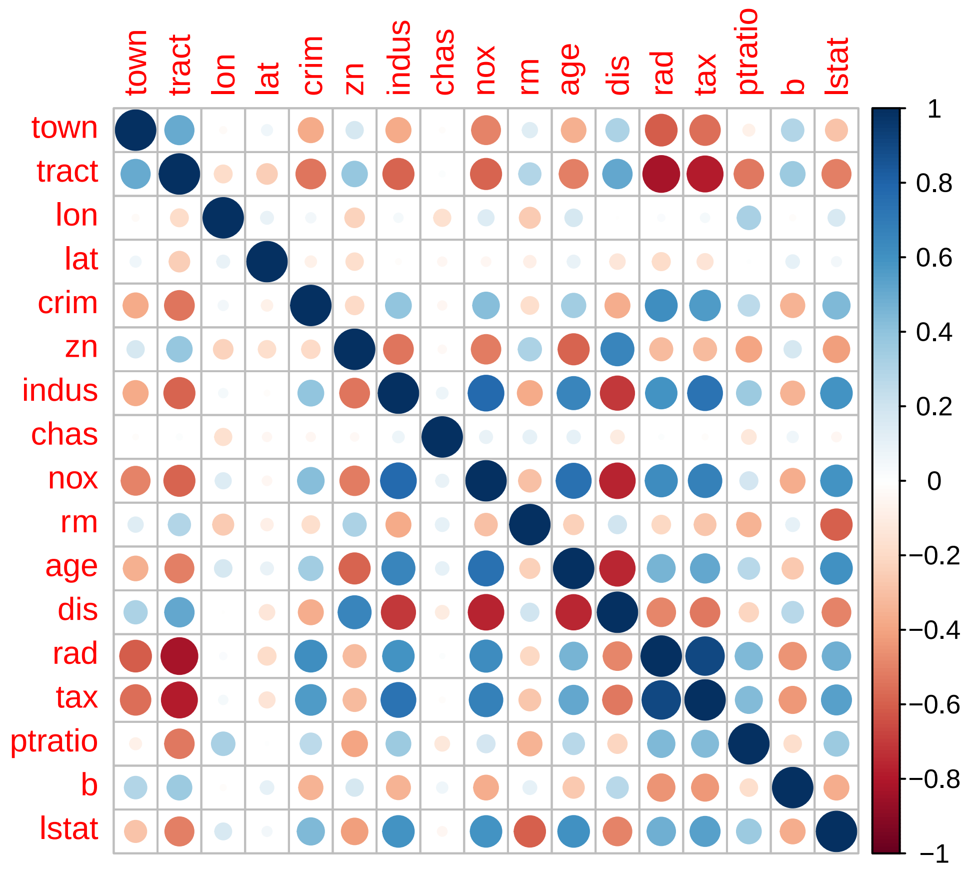
Above summaries hint to the following tranformations:

- To reduce the effect of different scales: normalize.
- To reduce the effect of different distributions: standardize.
- To make the distributions Gaussian-like: apply a Box-Cox transformation assuming positive values.

0.4.3 Looking for Collinearity

Removing highly correlated features would lead to an improvement of the the accuracy of our model. We are able to spot in the attached correlation plot some deep-red dots.

0.4.3.1 Correlation Plot



0.4.3.2 Pruning Highly Correlated Features

A threshold of 0.75 is established, above that the features will be removed by the following piece of code:

```
set.seed(13)
cutoff <- 0.75
correlations <- cor(train_set[,2:18])
highlyCorrelated <- findCorrelation(correlations, cutoff=cutoff)
for (value in highlyCorrelated) {
  print(names(train_set)[value])
}
```

```
[1] "rad"
[1] "town"
[1] "chas"
[1] "rm"
```

The new dimensions of our data set:

```
[1] 407  14
```

0.5 In search of the Right Algorithm

Summaries have done their work; however,the algorithm that will work in this case is still unknown. The task is spot-check several algorithms

0.5.1 Test Harness Design

- Based on above findings, we will center, scale, and apply a Box-Cox transformation to the data set. The aim is for all attributes to have a mean value of 0 and a standard deviation of 1.
- Test configuration: 10 cross-validation with 3 repeats

```
trainControl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)#Standard approach
metric <- "RMSE"
```

- A combination of linear and non-linear algorithm suited for regression are displayed in the following piece of code. All of them use default tuning parameters except CART which uses 3 supplementary parameters.

```
#lm
set.seed(13)
fit.lm <- train(cmedv~., data=train_set_1, method="lm", metric=metric, preProc=c("center",
"scale", "BoxCox"), trControl=trainControl)
#glm
set.seed(13)
fit.glm <- train(cmedv~., data=train_set_1, method="glm", metric=metric, preProc=c("center",
"scale", "BoxCox"), trControl=trainControl)
#rpart
set.seed(13)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.rpart<- train(cmedv~., data=train_set_1, method="rpart", metric=metric, tuneGrid = grid, preProc=c(
"scale", "BoxCox"), trControl=trainControl)
#svm
set.seed(13)
fit.svm <- train(cmedv~., data=train_set_1, method="svmRadial", metric=metric, preProc=c("center",
"scale", "BoxCox"), trControl=trainControl)
#KNN
set.seed(13)
fit.knn <- train(cmedv~., data=train_set_1, method="knn", metric=metric, preProc=c("center",
"scale", "BoxCox"), trControl=trainControl)
#GBM
set.seed(13)
fit.gbm <- train(cmedv~., data=train_set_1, method="gbm", metric=metric, preProc=c("center",
"scale", "BoxCox"), trControl=trainControl, verbose = FALSE)
#RF
set.seed(13)
fit.rf <- train(cmedv~., data=train_set_1, method="rf", metric=metric, preProc=c("center",
"scale", "BoxCox"),trControl=trainControl)
#Cubist
set.seed(13)
fit.cubist <- train(cmedv~., data=train_set_1, method="cubist", metric=metric,
preProc=c("center","scale", "BoxCox"), trControl=trainControl)
results <- resamples(list(LM=fit.lm, GLM=fit.glm, CART=fit.rpart, SVM=fit.svm, KNN = fit.knn,
                          GBM=fit.gbm, RF=fit.rf, CUBIST=fit.cubist))
```

Metrics used to estimate performance:

MAE or mean absolute error is a measure of difference between two continuous variable. As the name implies, it is an average of the absolute errors.

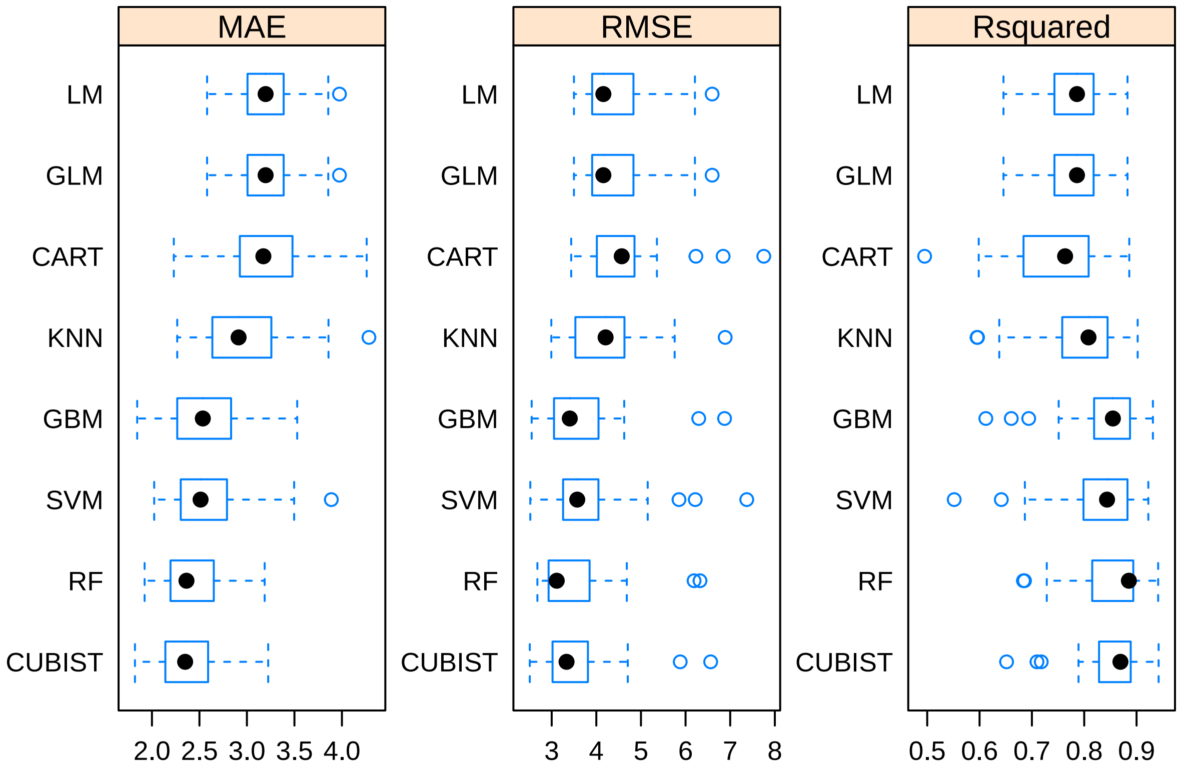
RMSE or root mean square error is the standard deviation of the residuals (prediction errors).

Rsquared coefficient of determination provides a goodness-of-measure for the predictions of the observations, a value between 0 and 1.

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

Models: LM, GLM, CART, SVM, KNN, GBM, RF, CUBIST
Number of resamples: 30

MAE
           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
LM     2.580596 3.008313 3.196660 3.219300 3.374664 3.974130    0
GLM    2.580596 3.008313 3.196660 3.219300 3.374664 3.974130    0
CART   2.230150 2.933621 3.173232 3.194254 3.477308 4.260553    0
SVM    2.023685 2.309961 2.512208 2.597353 2.787061 3.888508    0
KNN    2.266190 2.667125 2.912208 2.974813 3.238049 4.284390    0
GBM    1.844963 2.266562 2.535557 2.554524 2.825906 3.529168    0
RF     1.922915 2.197204 2.363437 2.440796 2.639479 3.186573    0
CUBIST 1.821189 2.145207 2.349444 2.408550 2.590341 3.223762    0

RMSE
           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
LM     3.498628 3.930644 4.158071 4.398940 4.751559 6.595659    0
GLM    3.498628 3.930644 4.158071 4.398940 4.751559 6.595659    0
CART   3.437145 4.026682 4.569959 4.637526 4.851867 7.752711    0
SVM    2.520364 3.271772 3.572795 3.910698 4.045353 7.371419    0
KNN    2.991510 3.532694 4.208150 4.228463 4.610205 6.887878    0
GBM    2.551058 3.052521 3.405114 3.678438 4.038493 6.874095    0
RF     2.679382 2.937749 3.113155 3.481208 3.847230 6.323407    0
CUBIST 2.508961 3.025181 3.332795 3.543771 3.811311 6.562324    0

Rsquared
            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
LM     0.6457292 0.7453515 0.7861744 0.7773151 0.8171288 0.8826862    0
GLM    0.6457292 0.7453515 0.7861744 0.7773151 0.8171288 0.8826862    0
CART   0.4950350 0.6898967 0.7634713 0.7459771 0.8063981 0.8860765    0
SVM    0.5515878 0.7991512 0.8435931 0.8220637 0.8811693 0.9224850    0
KNN    0.5952276 0.7592111 0.8081779 0.7890147 0.8421079 0.9021085    0
GBM    0.6120205 0.8196720 0.8547726 0.8383713 0.8873299 0.9312504    0
RF     0.6838403 0.8192519 0.8853370 0.8561005 0.8932170 0.9412959    0
CUBIST 0.6516223 0.8285645 0.8692101 0.8513204 0.8885263 0.9421445    0
```

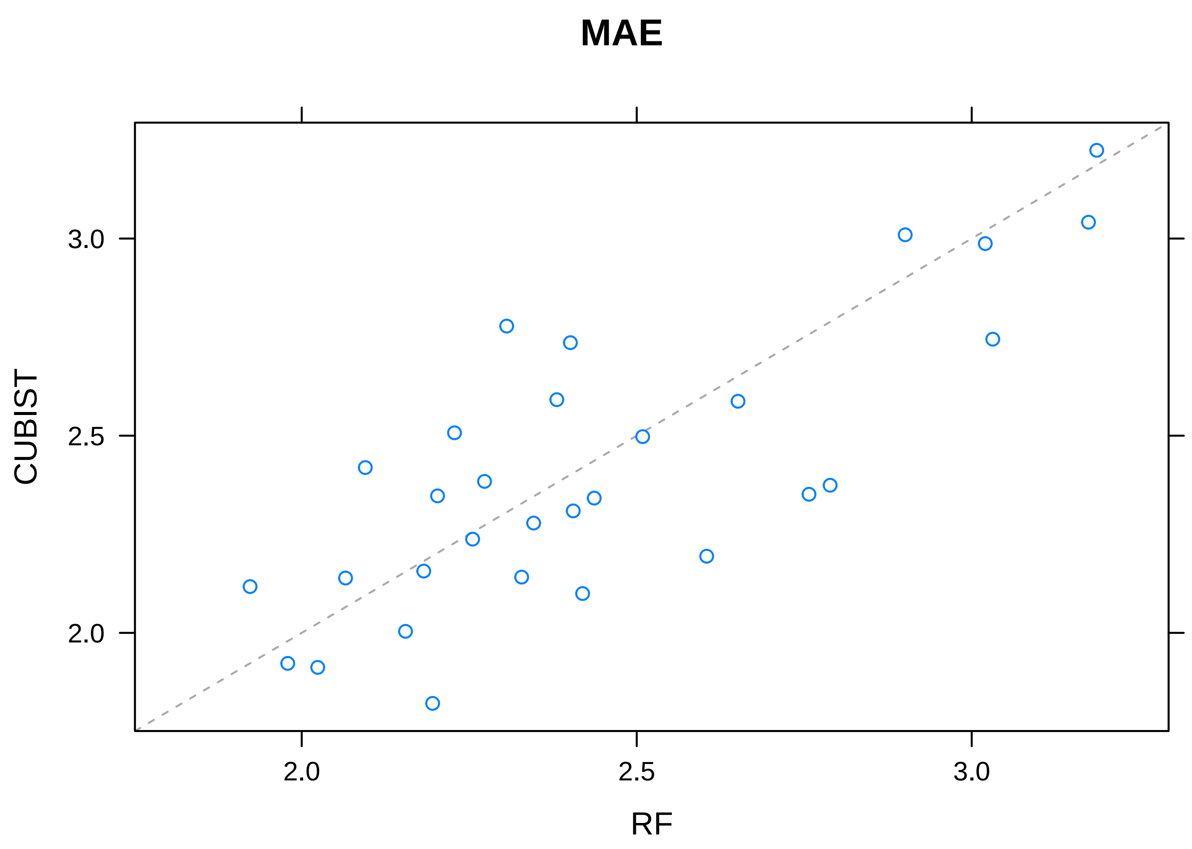
0.5.2 Visualizing the results

We use a dotplot to show a 95% Confidence Intervals for estimated metrics



It looks like non-linear algorithms exhibit better performance than the linear ones. Among them the RF and Cubist have the lowest RMSE.

As a precaution, let us check that our selected models are not strongly correlated to each other.



0.6 Improving the Selected Models

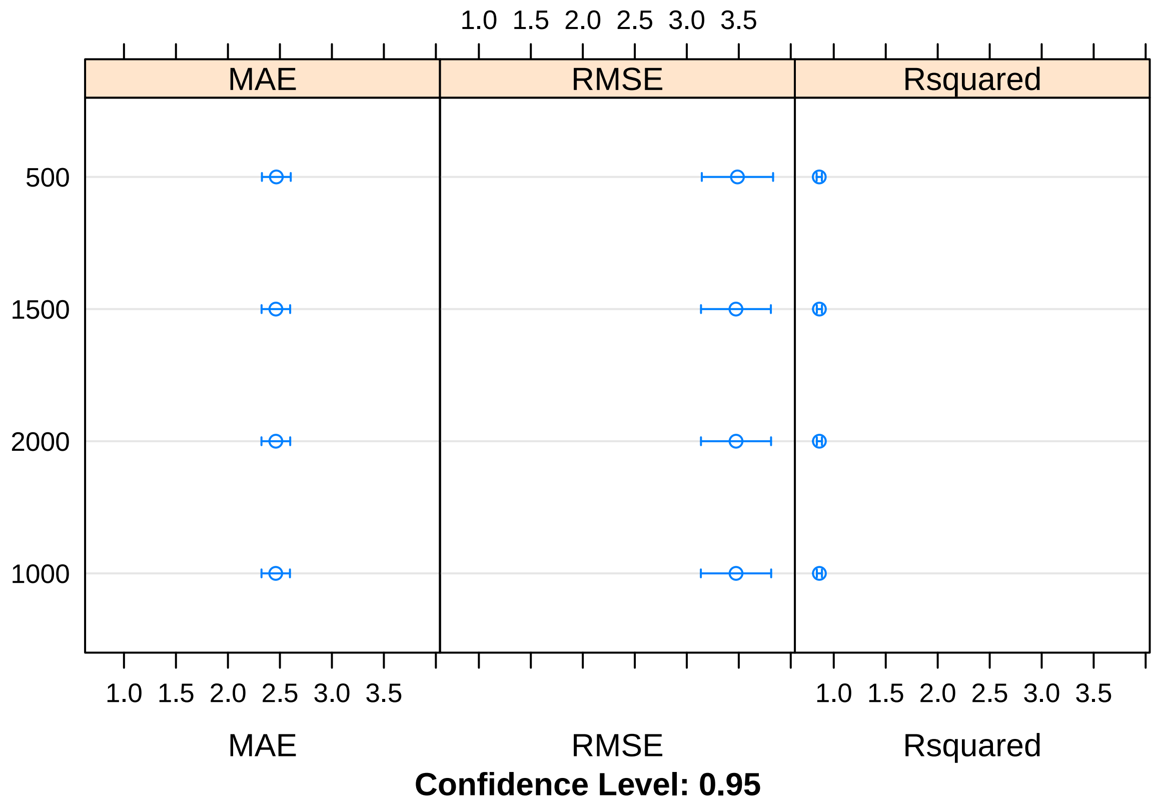
0.6.1 Tuning the Random Forest Algorithm

We undertake a manual search to determine the optimal number of trees and mtry. Code is borrowed from indicated site.[3]

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

Models: 500, 1000, 1500, 2000
Number of resamples: 30

MAE
         Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
500  1.934742 2.208485 2.390969 2.465449 2.655078 3.394381    0
1000 1.930199 2.206313 2.377079 2.459634 2.633820 3.413629    0
1500 1.939820 2.191139 2.371261 2.461172 2.642929 3.427533    0
2000 1.945104 2.188643 2.362293 2.460769 2.656239 3.418786    0

RMSE
         Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
500  2.517585 2.852976 3.188899 3.487205 3.904064 6.381007    0
1000 2.543038 2.843793 3.126820 3.473492 3.862886 6.396542    0
```

[3]htpps://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/

```
1500 2.538549 2.847850 3.138381 3.472709 3.853041 6.398130    0
2000 2.545236 2.844665 3.137312 3.473503 3.869886 6.360669    0

Rsquared
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
500  0.6789213 0.8454467 0.8854293 0.8604738 0.9059720 0.9343368    0
1000 0.6769558 0.8494699 0.8881023 0.8618106 0.9033264 0.9375204    0
1500 0.6755844 0.8491337 0.8871671 0.8617512 0.9031813 0.9379234    0
2000 0.6811698 0.8477958 0.8874045 0.8617028 0.9041132 0.9382278    0
```



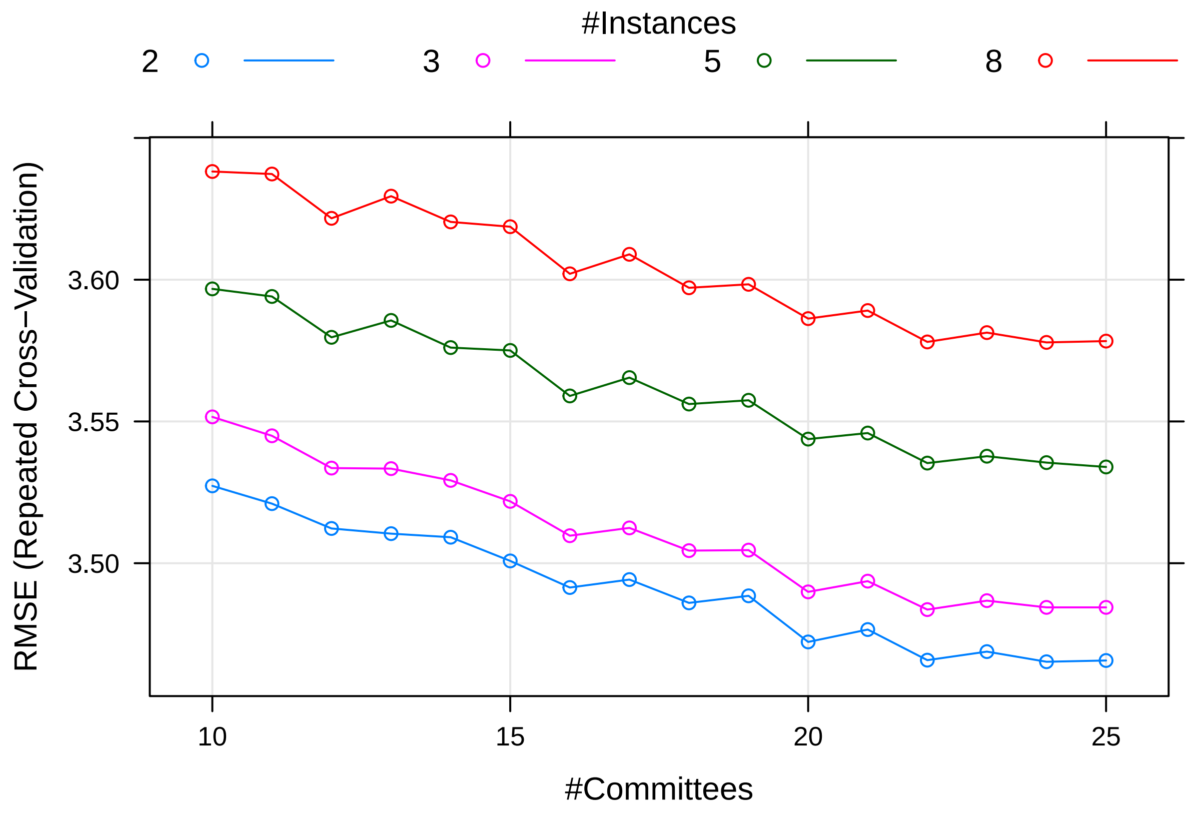
0.6.2 Tune the Cubist Algorithm

The cubist model uses a scheme called committees where iterative models trees are created. The final prediction is the resulting average of the predictions from each model tree. We have selected a short Fibonacci series to satisfy the "neighbors" parameter.

```
Cubist

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: 367, 367, 366, 366, 366, 366, ...
Resampling results across tuning parameters:

  committees  neighbors  RMSE      Rsquared   MAE
```

10	2	3.527282	0.8556195	2.412940
10	3	3.551613	0.8526988	2.403019
10	5	3.596763	0.8467252	2.412781
10	8	3.638172	0.8421039	2.424468
11	2	3.521009	0.8557910	2.405552
11	3	3.544936	0.8528736	2.397524
11	5	3.594081	0.8465065	2.406312
11	8	3.637281	0.8416311	2.415608
12	2	3.512255	0.8565623	2.398593
12	3	3.533553	0.8539742	2.389975
12	5	3.579684	0.8479749	2.401488
12	8	3.621651	0.8433311	2.409406
13	2	3.510422	0.8566852	2.398145
13	3	3.533359	0.8538981	2.388724
13	5	3.585641	0.8473062	2.403754
13	8	3.629472	0.8424184	2.410317
14	2	3.509163	0.8571337	2.399757
14	3	3.529216	0.8546003	2.388549
14	5	3.576055	0.8485037	2.401471
14	8	3.620393	0.8436224	2.408356
15	2	3.500808	0.8573854	2.397479
15	3	3.521813	0.8546925	2.391110
15	5	3.575055	0.8481991	2.405043
15	8	3.618694	0.8434101	2.410937
16	2	3.491419	0.8582545	2.394824
16	3	3.509699	0.8559282	2.386421
16	5	3.559005	0.8498981	2.400182
16	8	3.602079	0.8452526	2.406452
17	2	3.494228	0.8578491	2.396137
17	3	3.512453	0.8554383	2.389078
17	5	3.565466	0.8490812	2.404540
17	8	3.608951	0.8443872	2.409915
18	2	3.486000	0.8584314	2.397024
18	3	3.504435	0.8561451	2.390467
18	5	3.556142	0.8500532	2.405116
18	8	3.597147	0.8456728	2.408352
19	2	3.488489	0.8582392	2.398992
19	3	3.504618	0.8560187	2.393049
19	5	3.557464	0.8498059	2.409067
19	8	3.598372	0.8453177	2.411880
20	2	3.472233	0.8596884	2.394524
20	3	3.489886	0.8574622	2.388653
20	5	3.543771	0.8513204	2.408550
20	8	3.586284	0.8467644	2.412430
21	2	3.476601	0.8592851	2.399143
21	3	3.493672	0.8570600	2.391828
21	5	3.545909	0.8509466	2.410897
21	8	3.589124	0.8462041	2.414214
22	2	3.465801	0.8601578	2.394991
22	3	3.483641	0.8580041	2.388857
22	5	3.535313	0.8520337	2.407706
22	8	3.578053	0.8473679	2.410613
23	2	3.468810	0.8600617	2.397481
23	3	3.486788	0.8578590	2.390677

```
  23          5          3.537744  0.8518656  2.410770
  23          8          3.581338  0.8470825  2.414564
  24          2          3.465231  0.8603359  2.397064
  24          3          3.484420  0.8580887  2.391737
  24          5          3.535484  0.8521193  2.411523
  24          8          3.577888  0.8474673  2.414616
  25          2          3.465677  0.8602766  2.398868
  25          3          3.484414  0.8580256  2.391496
  25          5          3.533947  0.8521292  2.410437
  25          8          3.578348  0.8472198  2.416364

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were committees = 24 and neighbors = 2.
```



The results are self-explanatory: the optimal combination occurs at committee 24, neighbor 2.

0.7 Final Model

Based on above findings, the cubist model better captures the data features; we are ready to train our model

0.7.1 Train the Final Model

```r
set.seed(13)
x <- train_set_1[,2:14]
y <- train_set_1[,1]
preprocessed_data <- preProcess(x, method=c("center", "scale", "BoxCox"))
```

```
x_preprocessed <- predict(preprocessed_data, x)
finalModel <- cubist(x=x_preprocessed, y=y, committees=24)
summary(finalModel)
```

We perform same transformation on the test set as the ones applied to our train set, and reassure ourselves that both sets have the same shape

```
# transform the validation dataset
set.seed(13)
test_set_1 <- test_set %>% mutate(chas = as.numeric(as.character(chas)),
                                  town = as.numeric(as.factor(town))) %>%
  select(-medv, -rad, -town, -chas, -rm) %>% select(cmedv, everything())
new_df <- setdiff(train_set_1, test_set_1)
new_df
```

0.7.2 Making Predictions on the Test Dataset

```
test_set_x <- test_set_1[,2:14]
test_set_y <- test_set_1[,1]
test_set_x_preprocessed <- predict(preprocessed_data, test_set_x)
# use final model to make predictions on the validation dataset
predictions <- predict(finalModel, newdata=test_set_x_preprocessed, neighbors=2)
# calculate Metrics
rmse <- RMSE(predictions, test_set_y)
r2 <- R2(predictions, test_set_y)
```

Rsquared = 0.8073963

RMSE = 4.140558

0.8 Conclusions

Let us say that R2 is moderately satisfactory, but RMSE is on the high side, some steps to improve the model.

- With this particualr data, non-Linear models perform better than linear ones.A diversity of non-linear algorithms should be tested focusing in boosting-like models characterized by iterative tree creation.
- Additional approaches could be taking, for example: stacking. Meaning blending the predictions of multiples well performing models.
- Utilize more sophisticated transformations as Yeo-Johnson power transformation which is not sensitive to negative values.

0.9 Works Cited

0.9.1 Data

Newman, D.J. & Hettich, S. & Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

0.9.2 books

Mangiafico, S.S. 2016. Summary and Analysis of Extension Program Evaluation in R, version 1.18.1. rcompanion.org/handbook/. (Pdf version: rcompanion.org/documents/RHandbookProgramEvaluation.pdf.)

Brownlee, Jason. 2017. Machine Learning Mastery With R. version v1.6.(Pdf version: Machine Learning Mastery With R.pdf)

0.9.3 R packages

caret Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan and Tyler Hunt. (2019). caret: Classification and Regression Training. R package version 6.0-84. https://CRAN.R-project.org/package=caret

corrplot Taiyun Wei and Viliam Simko (2017). R package "corrplot": Visualization of a Correlation Matrix (Version 0.84). Available from https://github.com/taiyun/corrplot

Cubist Max Kuhn and Ross Quinlan (2018). Cubist: Rule- And Instance-Based Regression Modeling. R package version 0.2.2. https://CRAN.R-project.org/package=Cubist

gbm Brandon Greenwell, Bradley Boehmke, Jay Cunningham and GBM Developers (2019). gbm: Generalized Boosted Regression Models. R package version 2.1.5. https://CRAN.R-project.org/package=gbm

kableExtra Hao Zhu (2019). kableExtra: Construct Complex Table with 'kable' and Pipe Syntax. R package version 1.1.0. https://CRAN.R-project.org/package=kableExtra

kernlab Alexandros Karatzoglou, Alex Smola, Kurt Hornik, Achim Zeileis (2004). kernlab - An S4 Package for Kernel Methods in R. Journal of Statistical Software 11(9), 1-20. URL http://www.jstatsoft.org/v11/i09/

knitr Yihui Xie (2019). knitr: A General-Purpose Package for Dynamic Report Generation in R. R package version 1.23.

Yihui Xie (2015) Dynamic Documents with R and knitr. 2nd edition. Chapman and Hall/CRC. ISBN 978-1498716963

Yihui Xie (2014) knitr: A Comprehensive Tool for Reproducible Research in R. In Victoria Stodden, Friedrich Leisch and Roger D. Peng, editors, Implementing Reproducible Computational Research. Chapman and Hall/CRC. ISBN 978-1466561595

mlbench Friedrich Leisch & Evgenia Dimitriadou (2010). mlbench: Machine Learning Benchmark Problems. R package version 2.1-1.

psych Revelle, W. (2018) psych: Procedures for Personality and Psychological Research, Northwestern University, Evanston, Illinois, USA, https://CRAN.R-project.org/package=psych Version = 1.8.12.

randomForest Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

rpart Terry Therneau and Beth Atkinson (2019). rpart: Recursive Partitioning and Regression Trees. R package version 4.1-15. https://CRAN.R-project.org/package=rpart