Classification

Solon Karapanagiotis

28 Jul 2016

Contents

The following packages need to installed: `caret`, `rpart`, `gbm`.

Introduction

The data were collected from students enrolled in an introductory statistics course at a large university in the US over a four year period. An opening course survey was administered to the students. The anonymous survey was available on the course website and contained questions on student demographic variables, such as gender, height, eye color, whether or not a student exercises and for how many hours per week, etc. After seven semesters, the full data set contains 2,068 records on 14 different categorical and quantitative variables (Froelich and Stephenson 2013).

```
str(eyecolorgenderdata)
```

```
## 'data.frame':    2068 obs. of  14 variables:
##  $ gender       : Factor w/ 2 levels "female","male": 1 2 1 2 1 2 2 2 1 2 ...
##  $ age          : int  18 20 18 23 19 19 37 22 26 21 ...
##  $ year         : Factor w/ 6 levels "first","first\"",..: 1 6 1 3 5 5 6 6 3 6 ...
##  $ eyecolor     : Factor w/ 5 levels "blue","brown",..: 4 2 3 4 1 3 2 2 2 1 ...
##  $ height       : int  68 70 67 74 62 67 74 73 70 68 ...
##  $ miles        : num  195 120 200 140 60 0 511 210 120 90 ...
##  $ brothers     : int  0 3 0 1 0 0 1 3 2 1 ...
##  $ sisters      : int  1 0 1 1 1 1 1 2 1 1 ...
##  $ computertime : num  20 24 35 5 5 5 3 2 5 5 ...
##  $ exercise     : Factor w/ 2 levels "No","Yes": 2 1 2 2 2 2 1 2 1 2 ...
##  $ exercisehours: num  3 0 3 25 4 8 0 10 0 3 ...
##  $ musiccds     : int  75 50 53 50 30 100 50 100 130 34 ...
##  $ playgames    : num  6 0 8 0 2 0 3 6 0 0.5 ...
##  $ watchtv      : num  18 3 1 7 5 10 8 10 20 5 ...
```

The purpose of the analysis was to predict the gender of the student (classiffcation task) !!!

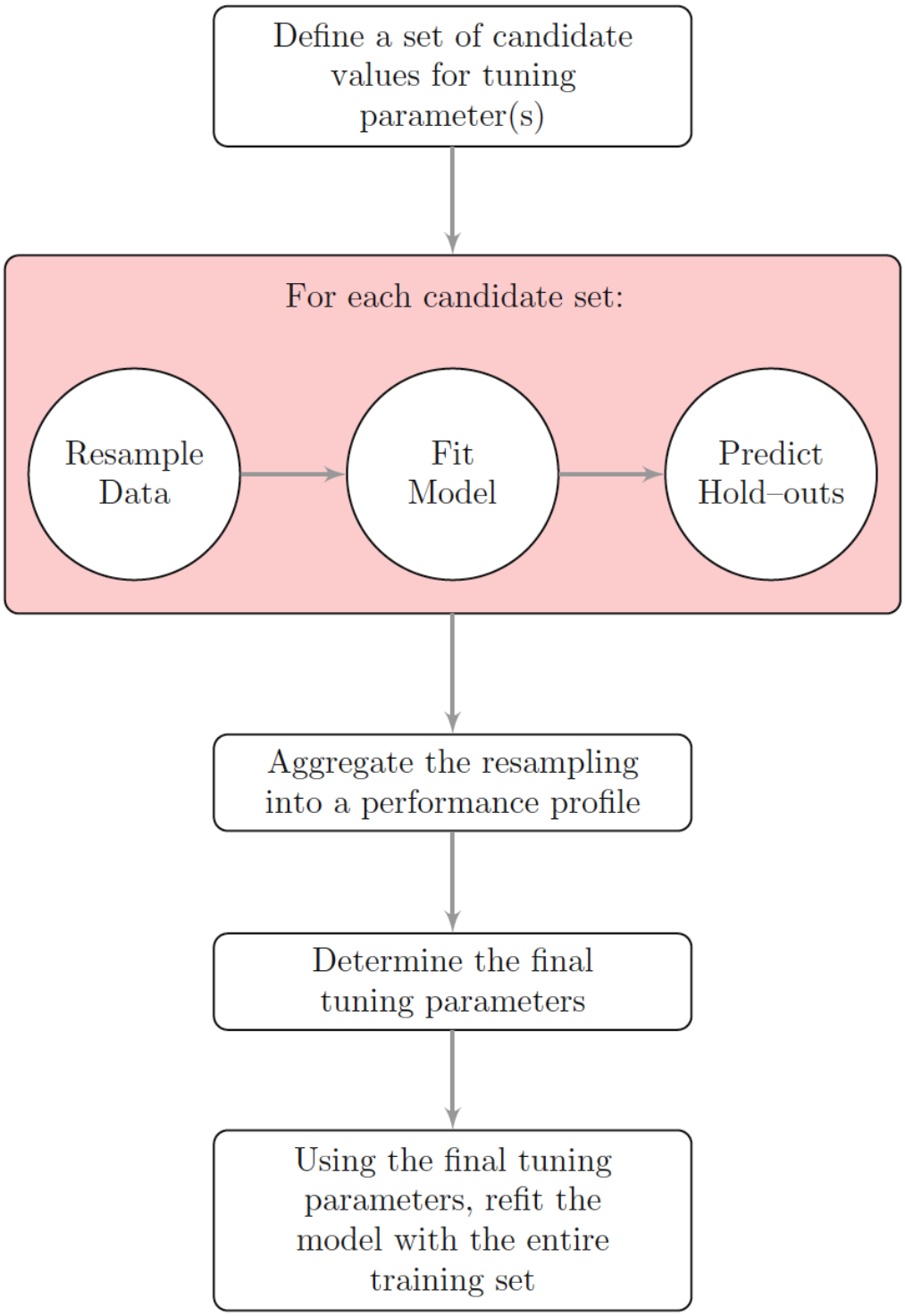
Methods

I set the sizes of training data (70%) and test data (30%), by simple random sampling from the dataset.

```
set.seed(186)
s <- sample(dim(eyecolorgenderdata)[1], 620)
test <- eyecolorgenderdata[s, ]
train <- eyecolorgenderdata[-s, ]
```

For the task I use Classifcation Trees as developed by Brieman et al. (1984) and Boosting as proposed by J. H. Friedman (2001).

Since both of the models contain one (or more) complexity parameters I chose to tune models by picking the complexity parameters that are associated with the optimal resampling statistics. Using resampling methods, such as the bootstrap or cross-validation, a set of modified data sets were created from the training samples. Each data set had a corresponding set of hold-out samples. For each candidate tuning parameter combination, a model was fit to each resampled data set and was used to predict the corresponding held out samples. The resampling performance was estimated by aggregating the results of each hold-out sample set. These performance estimates were used to evaluate which combination(s) of the tuning parameters were appropriate. Once the final tuning values were assigned, the final model was refit and fine-tuned using the entire training set (Kuhn 2008). The procedure is illustrated in the figure below (borrowed from (Kuhn 2008)). Finally, the prediction performance of each model was evaluated in the test set.

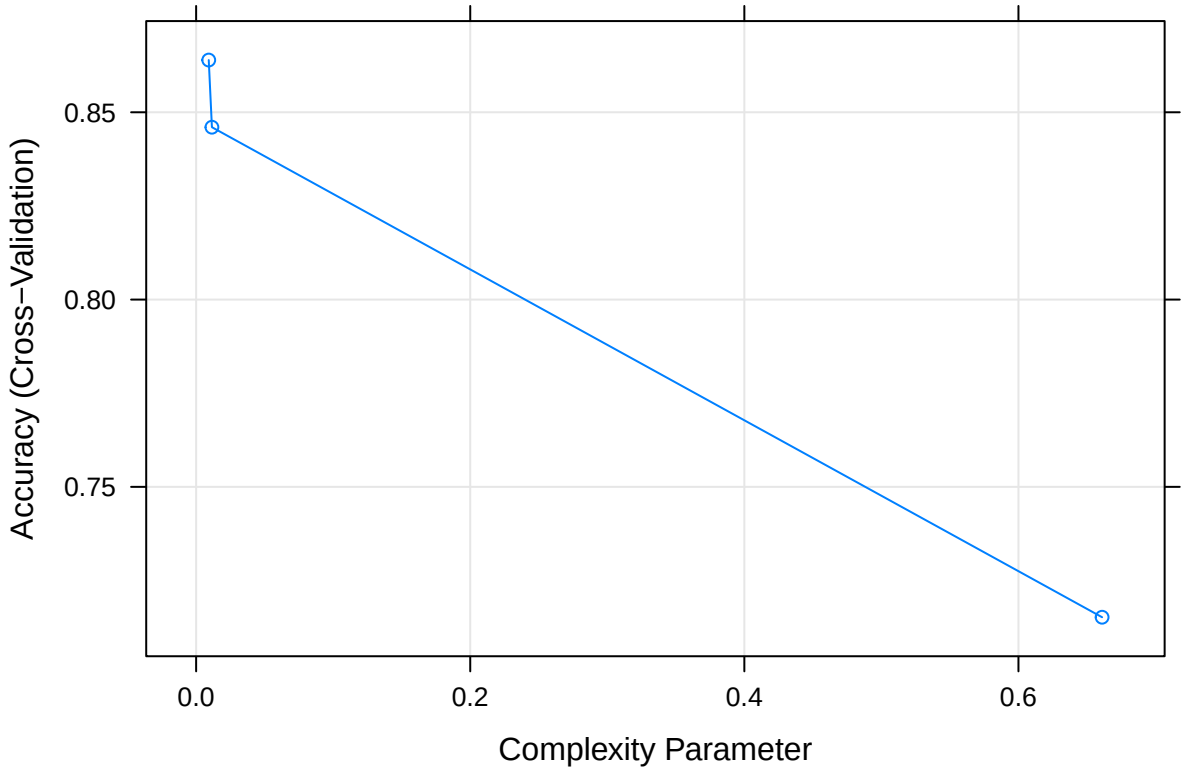

Define a set of candidate values for tuning parameter(s)
For each candidate set:
Resample Data
Fit Model
Predict Hold–outs
Aggregate the resampling into a performance profile
Determine the final tuning parameters
Using the final tuning parameters, refit the model with the entire training set

Classification task

Classification tree

The tuning parameter was the cost-complexity parameter (cp). The procedure described previously was used. For each value of cp the accuracy was calculated as the average accuracy of all the 10-fold cross validation samples of the training set. The cp value that corresponded to the higher accuracy was used.

```
set.seed(100)
rpartTune <- train(train[,-1], train[,1], method = "rpart", trControl = trainControl(method = "cv"))
plot(rpartTune)
```



- the `train()` function sets up a grid of tuning parameters for our tree model, fits each model and calculates a resampling based performance measure (I use the accuracy here),
- `method` stands for the chosen model, i.e., classification tree for us,
- `cv` stands for cross validation. Default is 10 fold cross-validation,
- the plot shows that based on the accuracy the optimal model was the one with cp = 0.0092.

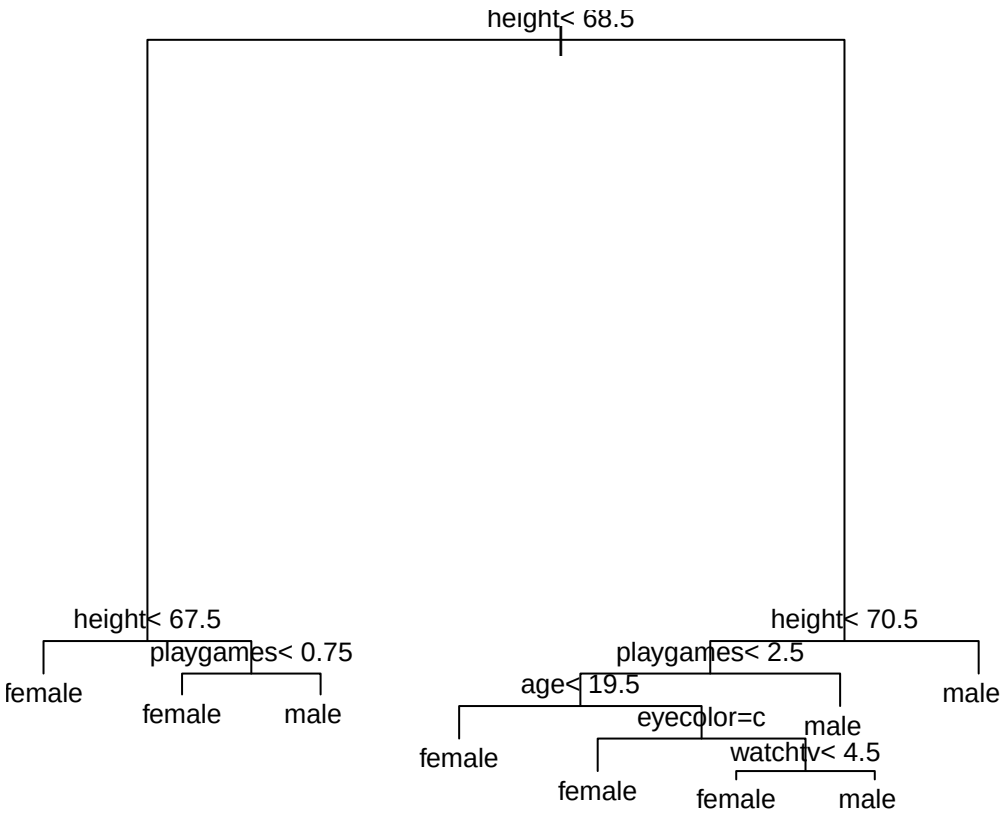
Now, I fit the model. The minimum number of observations that must exist in a node in order for a split to be attempted was chosen to be 15 (`minsplit` argument), which corresponds to 5 as the minimum number of observations in any terminal node. The splitting index was the Gini (`parms` argument).

```
set.seed(2479)
controlrpart <- rpart.control(minsplit = 15, cp = 0.001)
eye_rpart <- rpart(gender ~ . , data = train, control = controlrpart, method="class", parms = list(spli
```

I prune the tree based on the cp from the tuning process previously,

```
eye_rpart_prune <- prune(eye_rpart, cp = 0.009)
```

```
plot(eye_rpart_prune, compress = TRUE)
text(eye_rpart_prune, cex = 0.8)
```



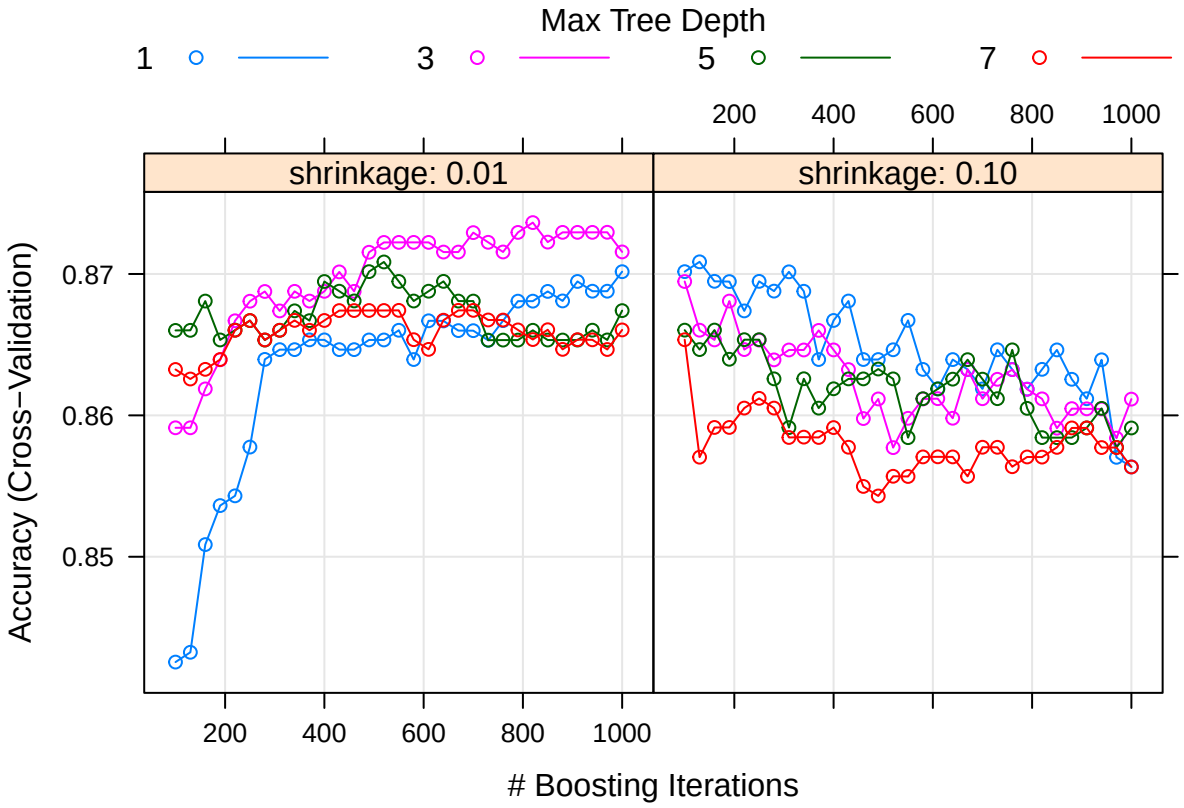
Let's take a look at our tree. A student with height less than 67.5 inches is predicted to be female (1st ternimal node from the left). If he/she is between 67.5 and 68.5 and plays video games for more than 0.75 hours per week he is predicted to be male (3rd ternimal node from the left).

Boosting

Stochastic Gradient Boosting is implemented (J. H. Friedman 2001). I tuned over the number of trees (i.e., boosting iterations), the complexity of the tree (indexed by interaction depth) and the learning rate (also known as shrinkage). A tuning parameter grid was constructed where interaction depth ranged from 1 to 7, number of trees ranged from 100 to 1000, and shrinkage (0.01 or 0.1). The minimum number of observations in the trees terminal nodes was set to 5. The combination of parameters with the best accuracy value was chosen (based on 10-fold cross validation).

```
gbmGrid <- expand.grid(.interaction.depth = seq(1, 7, by = 2),
                       .n.trees = seq(100, 1000, by = 30),
                       .shrinkage = c(0.01, 0.1),
                       .n.minobsinnode=5)

set.seed(100)
gbmTune <- train(train[,-1], train[,1], method = "gbm", tuneGrid = gbmGrid, verbose = FALSE, trControl =
plot(gbmTune)
```



- the `train()` function works as before. I use `verbose=F` to avoid printing out progress and performance indicators. Now, the `tuneGrid` is the grid of parameters specified at the `gbmGrid` object. It will take a couple of minutes, depending on the machine you have.

The tuning process is depicted in the figure. The parameters chosen (highest accuracy) where number of trees = 820, interaction depth = 3 and, shrinkage = 0.01.

Now, I fit the model.

```
set.seed(246)
forGBM <- train
forGBM$gender <- ifelse(forGBM$gender == "male", 1, 0)
gbmModel <- gbm(gender ~ ., distribution = "bernoulli", data = forGBM, n.trees = 820, interaction.depth
```

- One complication when using gbm for classification is that it expects that the outcome is coded as 0/1. That is what the `ifelse()` function is doing.
- The arguments I need to specify:
  - Since I need to classify into categories (male, femlale) I use `distribution = "bernoulli"`,
  - the total number of trees to fit (`n.trees`), derived from the tuning process,
  - the complexity of the tree (`interaction.depth`), derived from the tuning process,
  - the minimum number of observations in the trees terminal nodes (`n.minobsinnode`), I use 5 as I did with the CART before,
  - the learning rate (`shrinkage`), derived from the tuning process,
  - number of cross-validation folds to perform (`cv.folds`), I choose 10 as before,
  - and the the fraction of the training set observations randomly selected to propose the next tree in the expansion (`bag.fraction`).This introduces randomnesses into the model fit. If bag.fraction<1 then running the same model twice will result in similar but different fits. gbm uses the R random number generator so set.seed can ensure that the model can be reconstructed.

Results

As measure of prediction performance I use the accuracy. Confidence intervals are obtained by a procedure first given in Clopper and Pearson (1934).

for the CART

```
# predict on test data
testPred <- predict(eye_rpart_prune, newdata = test, type = "class")
#also
confusionMatrix(testPred, test$gender)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction female male
##     female    271   51
##     male       58  240
##
##                Accuracy : 0.8242
##                  95% CI : (0.7919, 0.8534)
##     No Information Rate : 0.5306
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6476
##  Mcnemar's Test P-Value : 0.5655
##
##             Sensitivity : 0.8237
##             Specificity : 0.8247
##          Pos Pred Value : 0.8416
##          Neg Pred Value : 0.8054
##              Prevalence : 0.5306
##          Detection Rate : 0.4371
##    Detection Prevalence : 0.5194
##       Balanced Accuracy : 0.8242
##
##        'Positive' Class : female
##
```

The prediction accuracy was estimated as 0.8242 with 95% CI (0.7919, 0.8534).

for boosting

```
gbmPred <- predict(gbmModel, newdata = test, type = "response",
## The number of trees must be explicitly set,
n.trees = 820)
gbmClass <- ifelse(gbmPred > .5, "male", "female")
gbmClass <- factor(gbmClass, levels = levels(train$gender))
confusionMatrix(gbmClass, test$gender)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction female male
##     female    279   49
##     male       50  242
##
##                Accuracy : 0.8403
##                  95% CI : (0.8091, 0.8683)
##     No Information Rate : 0.5306
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6795
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8480
##             Specificity : 0.8316
##          Pos Pred Value : 0.8506
##          Neg Pred Value : 0.8288
##              Prevalence : 0.5306
##          Detection Rate : 0.4500
##    Detection Prevalence : 0.5290
##       Balanced Accuracy : 0.8398
##
##        'Positive' Class : female
##
```

- The prediction function for this model does not predict the winning class. Using `predict(gbmModel, type = "response")` calculate the class probability for the class encoded as a 1 (in our example, a male student).
- with the `ifelse()` function I convert the probability to a factor variable (male, female).

The prediction accuracy was estimated as 0.8403 with 95% CI (0.8091, 0.8683).

Discussion

Starting from the classification tree, it does not produce optimal predictive performance (82% on average) even though "optimal" depends on the context of the problem we are working on. Hence, in order to improve the result boosting was implemented. Since classification trees are a low bias/high variance technique, the ensemble of trees (boosting) helps to drive down variance, producing a result that has low bias and low variance (shorter CI). Nevertheless, in this dataset the predictive performance increased marginally (from 82% to 84%). As a result, the classification tree seems more attractive option for our final model since it is highly interpretable and easy to implement. A further note for the boosting algorithm regards the tuning parameters. Even though the search performed was quite extensive I have to acknowledge that a more thorough search could have given different combinations and potentially could have provided with better results. Still we are confident on the result since it seems for these data, that boosting can find an optimal setting fairly quickly. Boosting shows (see figure) that as the number of trees increases, model performance improves for low values of shrinkage and degrades for higher values of shrinkage. But, whether a lower or higher value of shrinkage is selected, their difference in predictive performance is very small (resutls not shown).

References

Brieman, L, JH Friedman, R Olshen, and C Stone. 1984. "Classification and Regression Trees (Belmont, cA: Wadsworth)." Inc.

Clopper, Charles J, and Egon S Pearson. 1934. "The Use of Confidence or Fiducial Limits Illustrated in the Case of the Binomial." *Biometrika* 26 (4). JSTOR: 404–13.

Friedman, Jerome H. 2001. "Greedy Function Approximation: A Gradient Boosting Machine." *Annals of Statistics*. JSTOR, 1189–1232.

Froelich, Amy G, and W Robert Stephenson. 2013. "Does Eye Color Depend on Gender? It Might Depend on Who or How You Ask." *Journal of Statistics Education* 21 (2).

Kuhn, Max. 2008. "Caret Package." *Journal of Statistical Software* 28 (5).