MLStats: Tree Based Methods

Darshan Patel

2/27/2019

In this assignment, mimic the lab exercises from ISLR Chapter 8: Tree-Based Methods.

Libraries

Load the following libraries.

```
rm(list = ls())
library(MASS)
library(knitr)
library(tidyverse)
```

```
## Warning: package 'tibble' was built under R version 3.4.4
## Warning: package 'tidyr' was built under R version 3.4.4
## Warning: package 'purrr' was built under R version 3.4.4
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
library(tree)
```

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

```
library(randomForest)
```

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

```
library(gbm)
```

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

```
library(glmnet)
```

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

```
library(class)
```

Dataset

In this assignment, the dataset that will be used is `glass.txt`. It contains a number of chemical compositions of glass. The goal is to properly classify types of glass based on its composition. This type of information is good for forensics use.

(Source: https://archive.ics.uci.edu/ml/datasets/glass+identification)

```
df = read_delim("glass.txt", delim = ',', col_names = FALSE)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
```

```
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double(),
##   X11 = col_integer()
## )
```

```
colnames(df) = c("ID", "RI", "Na", "Mg", "Al",
                 "Si", "K", "Ca", "Ba", "Fe", "type")
df = df %>% subset(select = -ID)
df$type = as.factor(df$type)
```

The size of the dataset is

```
nrow(df)
```

```
## [1] 214
```

The number of variables in the dataset is

```
ncol(df)
```

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

The variables are listed below:

```
colnames(df)
```

```
##  [1] "RI"   "Na"   "Mg"   "Al"   "Si"   "K"    "Ca"   "Ba"   "Fe"   "type"
```

Basic Tree Model

First, create a train and test split for the data.

```
set.seed(2019)
indices = sample(1:nrow(df), size = 0.7*nrow(df))
train = df[indices,]
test = df[-indices,]
```

Regress on `type` using the training data.

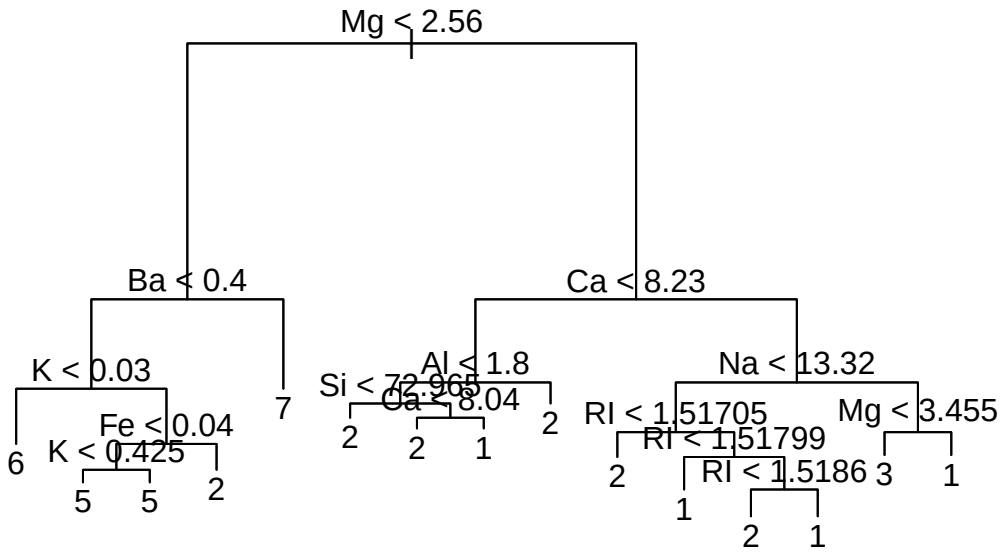
```
model_base = tree(type~., data = train)
summary(model_base)
```

```
##
## Classification tree:
## tree(formula = type ~ ., data = train)
## Number of terminal nodes:  15
## Residual mean deviance:  0.9569 = 128.2 / 134
## Misclassification error rate: 0.1812 = 27 / 149
```

According to this, the tree has 15 terminal nodes and a residual mean deviance of 0.95.

Plot the tree.

```
plot(model_base)
text(model_base, pretty = 0)
```



It can be seen that whether a magesium composition was less than 2.56% was most important in predicting `type`, followed closely by calcium and barium composition. Let's look at it more analytically.

```
model_base
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 149 450.400 2 ( 0.32886 0.34228 0.08054 0.04698 0.04698 0.15436 )
##     2) Mg < 2.56 43 109.600 7 ( 0.00000 0.20930 0.00000 0.16279 0.16279 0.46512 )
##       4) Ba < 0.4 24  62.020 2 ( 0.00000 0.33333 0.00000 0.29167 0.29167 0.08333 )
##         8) K < 0.03 8   6.028 6 ( 0.00000 0.12500 0.00000 0.00000 0.87500 0.00000 ) *
##         9) K > 0.03 16  31.460 2 ( 0.00000 0.43750 0.00000 0.43750 0.00000 0.12500 )
##          18) Fe < 0.04 11  19.970 5 ( 0.00000 0.18182 0.00000 0.63636 0.00000 0.18182 )
##            36) K < 0.425 5   6.730 5 ( 0.00000 0.40000 0.00000 0.60000 0.00000 0.00000 ) *
##            37) K > 0.425 6   7.638 5 ( 0.00000 0.00000 0.00000 0.66667 0.00000 0.33333 ) *
##          19) Fe > 0.04 5   0.000 2 ( 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 ) *
##       5) Ba > 0.4 19   7.835 7 ( 0.00000 0.05263 0.00000 0.00000 0.00000 0.94737 ) *
##     3) Mg > 2.56 106 227.100 1 ( 0.46226 0.39623 0.11321 0.00000 0.00000 0.02830 )
##       6) Ca < 8.23 34  46.100 2 ( 0.17647 0.76471 0.00000 0.00000 0.00000 0.05882 )
##        12) Al < 1.8 28  29.100 2 ( 0.21429 0.78571 0.00000 0.00000 0.00000 0.00000 )
##          24) Si < 72.965 17   7.606 2 ( 0.05882 0.94118 0.00000 0.00000 0.00000 0.00000 ) *
##          25) Si > 72.965 11  15.160 2 ( 0.45455 0.54545 0.00000 0.00000 0.00000 0.00000 )
##            50) Ca < 8.04 6   5.407 2 ( 0.16667 0.83333 0.00000 0.00000 0.00000 0.00000 ) *
##            51) Ca > 8.04 5   5.004 1 ( 0.80000 0.20000 0.00000 0.00000 0.00000 0.00000 ) *
##        13) Al > 1.8 6   7.638 2 ( 0.00000 0.66667 0.00000 0.00000 0.00000 0.33333 ) *
##       7) Ca > 8.23 72 144.000 1 ( 0.59722 0.22222 0.16667 0.00000 0.00000 0.01389 )
##        14) Na < 13.32 42  62.100 1 ( 0.64286 0.33333 0.02381 0.00000 0.00000 0.00000 )
##          28) RI < 1.51705 7  11.150 2 ( 0.14286 0.71429 0.14286 0.00000 0.00000 0.00000 ) *
##          29) RI > 1.51705 35  39.900 1 ( 0.74286 0.25714 0.00000 0.00000 0.00000 0.00000 )
##            58) RI < 1.51799 22   8.136 1 ( 0.95455 0.04545 0.00000 0.00000 0.00000 0.00000 ) *
##            59) RI > 1.51799 13  17.320 2 ( 0.38462 0.61538 0.00000 0.00000 0.00000 0.00000 )
##             118) RI < 1.5186 7   0.000 2 ( 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 ) *
##             119) RI > 1.5186 6   5.407 1 ( 0.83333 0.16667 0.00000 0.00000 0.00000 0.00000 ) *
```

```
##        15) Na > 13.32 30  59.820 1 ( 0.53333 0.06667 0.36667 0.00000 0.00000 0.03333 )
##          30) Mg < 3.455 9  22.910 3 ( 0.22222 0.22222 0.44444 0.00000 0.00000 0.11111 ) *
##          31) Mg > 3.455 21  26.730 1 ( 0.66667 0.00000 0.33333 0.00000 0.00000 0.00000 ) *
```

Looking at one of the terminal nodes, for example node 50, the split criterion is whether the calcium composition is less than 8.04% or not. There are 6 observations at this node with a deviation of 5.407. Furthermore, 16.67% of the observations are of glass type 2 while the other 83.33% are of another type (type 1 if looked at node 51).

Now find the test error rate.

```
predictions = predict(model_base, test, type = "class")
table(predictions, test$type)
```

```
##
## predictions  1  2  3  5  6  7
##           1 16  5  0  0  0  0
##           2  5 17  4  3  0  0
##           3  0  1  1  0  0  0
##           5  0  0  0  3  0  0
##           6  0  2  0  0  2  0
##           7  0  0  0  0  0  6
```

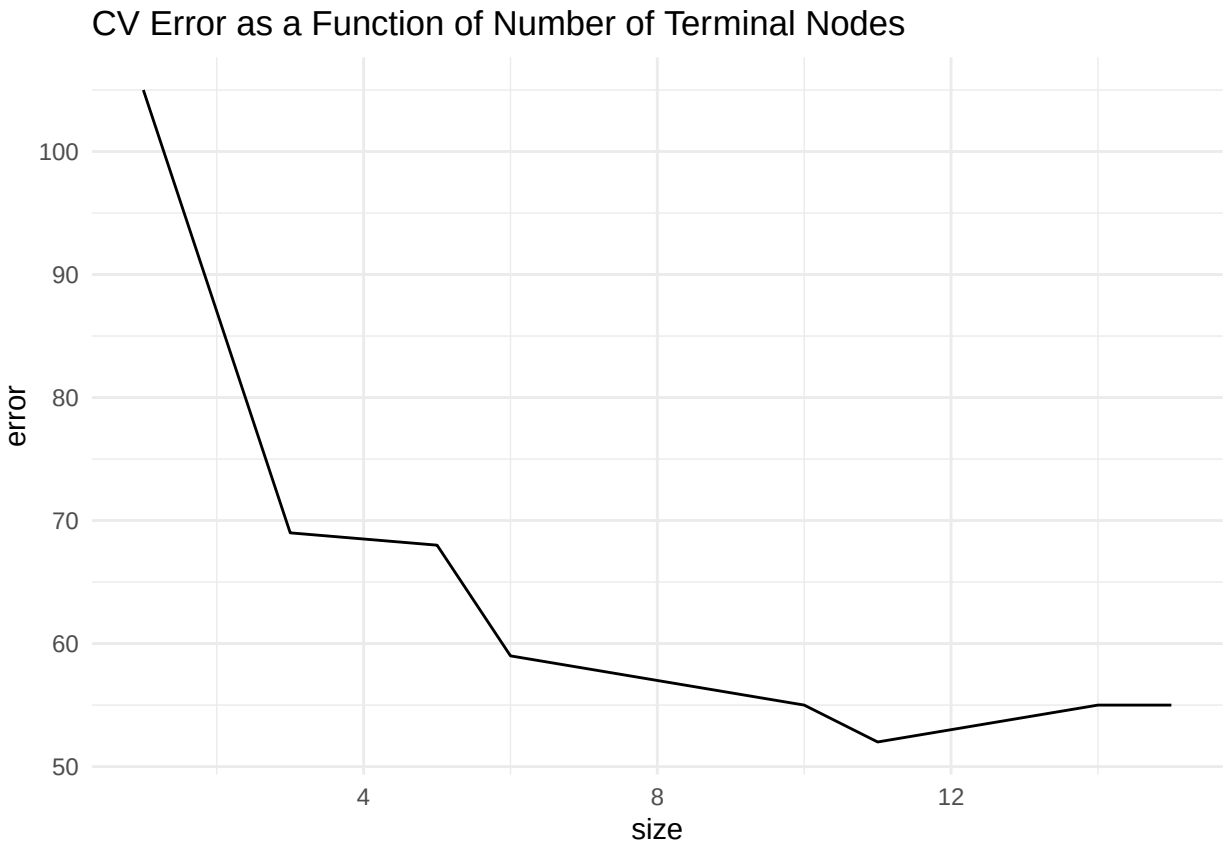
According to the confusion matrix, glass types of 5 and 7 were correctly identified 100% of the time. The test error rate is

```
error_base = (nrow(test) - sum(diag(table(predictions, test$type)))) / nrow(test)
error_base
```

```
## [1] 0.3076923
```

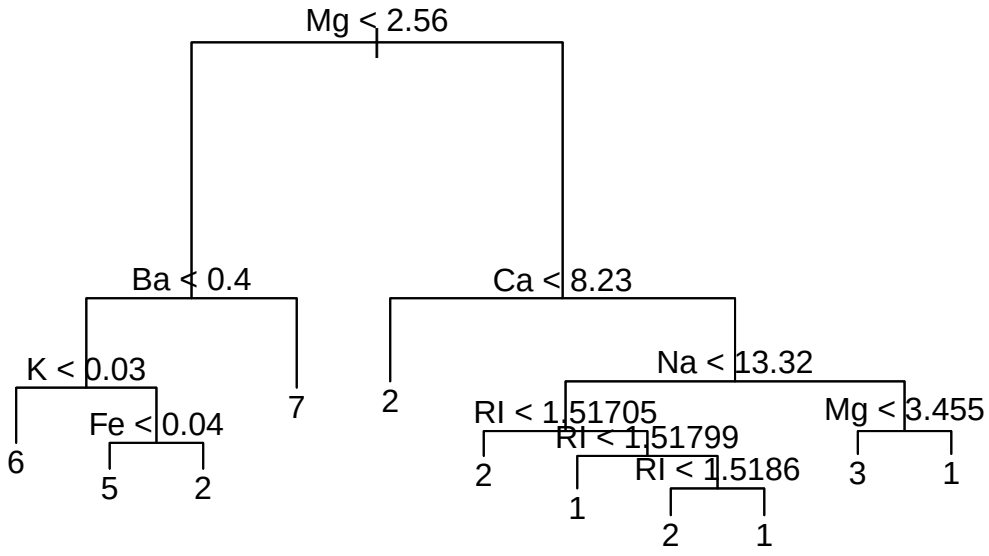
Can this be improved on by pruning the tree?

```
set.seed(3)
model_base_cv = cv.tree(model_base, FUN = prune.misclass)
model_base_cv_df = data.frame(size = model_base_cv$size,
                              error = model_base_cv$dev)
ggplot(model_base_cv_df, aes(x = size, y = error)) + geom_path() +
  ggtitle("CV Error as a Function of Number of Terminal Nodes") +
  theme_minimal()
```



The best number of terminal nodes to use is 11. This is less than the number of terminal nodes used in the tree above. Using this new value, a pruned tree is made.

```
model_base_pruned = prune.misclass(model_base, best = 11)
plot(model_base_pruned)
text(model_base_pruned, pretty = 0)
```



The important chemical elements used to classify `type` remain the same. The confusion matrix is

```
predictions = predict(model_base_pruned, test, type = "class")
table(predictions, test$type)
```

```
##
## predictions  1  2  3  5  6  7
##           1 15  5  0  0  0  0
##           2  6 17  4  3  0  0
##           3  0  1  1  0  0  0
##           5  0  0  0  3  0  0
##           6  0  2  0  0  2  0
##           7  0  0  0  0  0  6
```

According to the confusion matrix, there is no improvement when compared to the previous confusion matrix. The test error rate is

```
error_base_pruned = (nrow(test) - sum(diag(table(predictions, test$type)))) / nrow(test)
error_base_pruned
```

```
## [1] 0.3230769
```

The error rate has gone up after pruning! This shows that pruning does not help improve classification of glass type.

The next tree-based method to consider is bagging.

Bagging

Create a bagged model with $m = p$ variables to consider at each split.

```
model_bag = randomForest(type~., data = train, ntree = 100,
                         mtry = ncol(train)-1, importance = TRUE)
model_bag
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = train, ntree = 100, mtry = ncol(train) -      1, importance
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 9
##
##         OOB estimate of  error rate: 27.52%
## Confusion matrix:
##    1  2 3 5 6  7 class.error
## 1 36  9 3 0 0  1   0.2653061
## 2  9 38 2 1 0  1   0.2549020
## 3  7  3 2 0 0  0   0.8333333
## 5  0  2 0 5 0  0   0.2857143
## 6  0  0 0 0 7  0   0.0000000
## 7  2  1 0 0 0 20   0.1304348
```

When using 100 trees with 9 variables to consider at each split, the out of bag error is 27.52%. Furthermore, the test set error is

```
predictions = predict(model_bag, test, type = "class")
error_bag = (nrow(test) - sum(diag(table(predictions, test$type)))) / nrow(test)
error_bag
```

```
## [1] 0.2
```

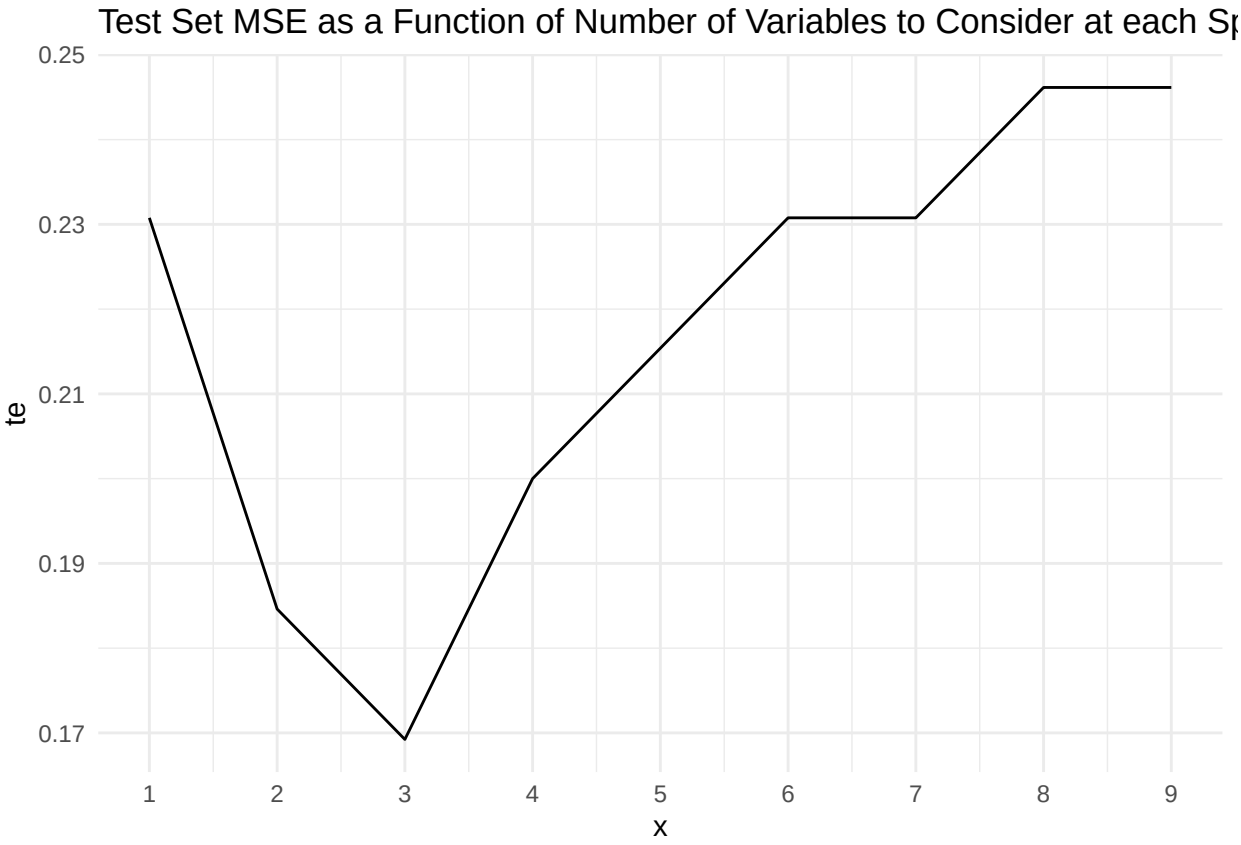
This error is less than that by using a single tree, unpruned and pruned. Using all the variables helped when bagged together, but is there a certain subset of variables that can give most of the information to predict

type? Try using random forests.

Random Forest

Determine which value of m, number of variable to consider at each split, results in the lowest test set MSE.

```
set.seed(100)
test_errors = c()
for(i in 1:9){
  model = randomForest(type~., data = train,
                       mtry = i, ntree = 100, importance = TRUE)
  predictions = predict(model, test, type = "class")
  test_errors = c(test_errors, (nrow(test) - sum(diag(table(predictions,
                                                            test$type)))) / nrow(test))
}
te_df = data.frame(x = 1:9, te = test_errors)
ggplot(te_df, aes(x = x, y = te)) + geom_path() +
  scale_x_continuous(breaks = 1:9) +
  ggtitle("Test Set MSE as a Function of Number of Variables to Consider at each Split") +
  theme_minimal()
```



The best number of variables to consider is $m = 3$. Using this parameter, create a random forest.

```
model_rf = randomForest(type~., data = train,
                        mtry = 3, ntree = 5,
                        importance = TRUE)
importance(model_rf)
```

```
##             1           2         3        5        6         7
## RI  5.1479331  1.78789523 0.6374602 1.118034 1.767767  1.118034
## Na -0.6235856  1.67238190 1.8132963 1.825742 1.767767  1.118034
## Mg  3.2433449  1.37331734 1.1180340 3.033899 1.118034  1.364928
## Al  1.3238920 -0.28523242 0.6826814 2.390457 1.118034  1.118034
## Si  1.1180340 -1.63760117 1.1180340 1.677051 0.000000 -1.118034
## K   4.8161021 -0.67141582 1.1180340 1.118034 2.041241  2.381060
## Ca  0.7629961  3.38484546 1.1180340 1.118034 1.118034  0.000000
## Ba  2.7361787  1.94968242 0.0000000 1.118034 0.000000  4.218944
## Fe  0.0000000 -0.07849477 0.0000000 0.000000 0.000000  0.000000
##    MeanDecreaseAccuracy MeanDecreaseGini
## RI            3.88375305        14.320297
## Na            2.09294202        14.041321
## Mg            3.73094450        14.134454
## Al            2.34337961        13.718664
## Si           -0.02668043         9.626032
## K             4.10681475        11.452191
## Ca            3.86940582        13.043578
## Ba            4.66703131        12.365840
## Fe           -0.02668043         6.570106
```

It appears to be that if reflective index was taken out of the model, then the average out of bag accuracy decreases by 2.9%. Similarly, if barium composition is dropped, then the average out of bag accuracy decreases by 3.3%. Furthermore, if reflective index was taken out of the model, the average decrease in node impurity is 20.75%.

The test error rate when using this model is

```
predictions = predict(model_rf, test, type = "class")
error_rf = (nrow(test) - sum(diag(table(predictions,
                                        test$type)))) / nrow(test)
error_rf
```

```
## [1] 0.1692308
```

This error rate is higher than the one for a bagged model. Subsetting on the variables did not help with improving the model.

The last approach to consider is boosting.

Boosting

The function `gbm` can only solve binary classification problems. Therefore, the `type` variable will be divided into two groups: `0` and `1`. In group `0`, the glass types will be building windows and vehicle windows. In group `1`, the glass types will be containers, tableware and headlamps.

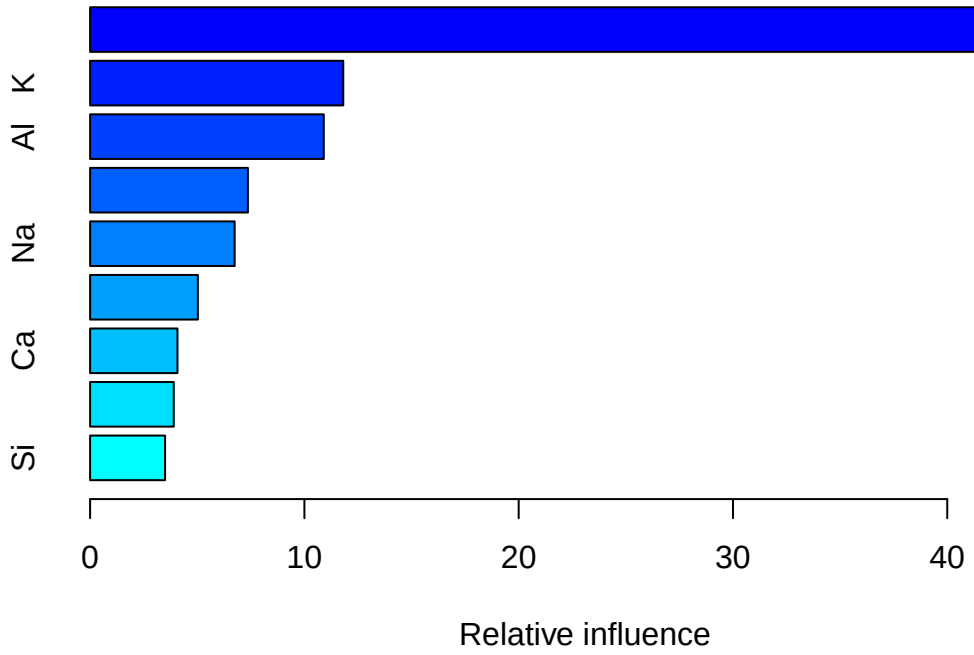
```
type_class = ifelse(as.integer(df$type) < 4, 0, 1)
new_df = cbind(df, typeclass = type_class) %>% subset(select = -type)
new_train = new_df[indices,]
new_test = new_df[-indices,]
```

Create a boosting model to predict `typeclass`.

```
model_boost = gbm(typeclass ~ ., data = new_train,
                  distribution = "bernoulli", n.trees = 100,
                  interaction.depth = 4)
```

Plot the importance of variables.

```
summary(model_boost)
```




```
##    var   rel.inf
## Mg  Mg 46.663782
## K    K 11.814070
## Al  Al 10.903883
## RI  RI  7.364969
## Na  Na  6.744822
## Fe  Fe  5.029600
## Ca  Ca  4.076605
## Ba  Ba  3.903945
## Si  Si  3.498323
```

It appears to be that magnesium concentration is heavily important, relatively compared to the other variables. Using this model, the test set error is

```
predictions = predict(model_boost, new_test,
                      n.trees = 100, type = "response")
probs = ifelse(predictions > 0.5, 1, 0)
error_boost = (nrow(new_test) - sum(diag(table(probs,
                                               new_test$typeclass)))) / nrow(new_test)
error_boost
```

```
## [1] 0.07692308
```

This test set error rate is low. However, it cannot be compared with the other error values since this dealt with a different classification problem. However, different classification algorithms can be used to judge this value.

KNN

Use KNN to predict `typeclass`.

```
set.seed(100)
model_knn = knn(new_train, new_test,
                cl = new_train$typeclass, k = 5)
predictions = ifelse(model_knn == 0, 0, 1)
error_knn = (nrow(new_test) - sum(diag(table(predictions,
                                             new_test$typeclass)))) / nrow(new_test)
error_knn
```

```
## [1] 0.01538462
```

The KNN approach did better than boosted trees. How about logistic regression?

Logistic Regression

Create a logistic regression model to predict `typeclass` and report the test error rate.

```
set.seed(100)
model_log = glm(typeclass ~., data = new_train, family = "binomial")
probs = predict(model_log, new_test, type = "response")
predictions = ifelse(probs == 0, 0, 1)
error_log = (nrow(new_test) - sum(diag(table(predictions,
                                             new_test$typeclass)))) / nrow(new_test)
error_log
```

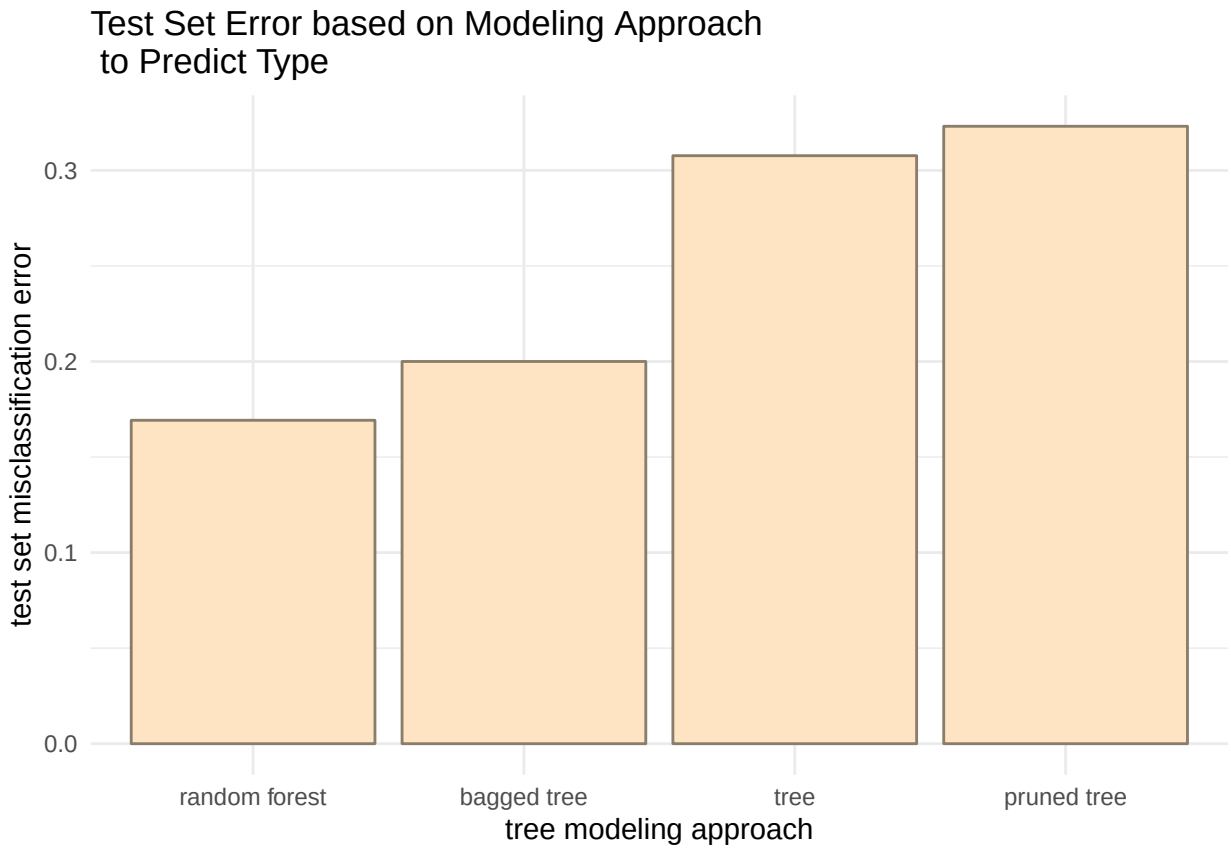
```
## [1] 0.2153846
```

The logistic regression approach performed the worst.

Conclusion

The test set errors for classification of `type` is plotted.

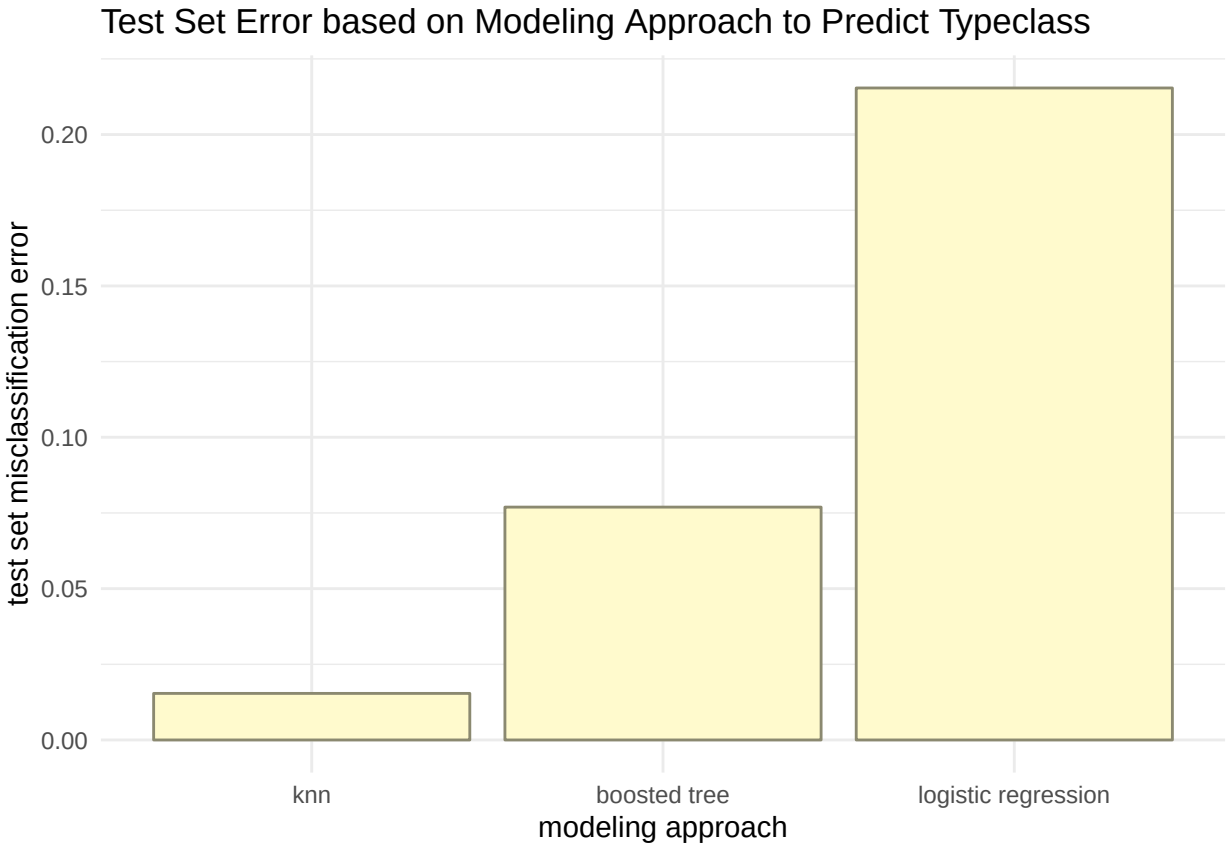
```
type_df = data.frame("model" = c("tree", "pruned tree",
                                 "bagged tree", "random forest"),
                     "error" = c(error_base, error_base_pruned,
                                 error_bag, error_rf))
ggplot(type_df, aes(x = reorder(model, error), y = error)) +
  geom_col(color = "bisque4", fill = "bisque1") +
  labs(x = "tree modeling approach", y = "test set misclassification error") +
  ggtitle("Test Set Error based on Modeling Approach \n to Predict Type") +
  theme_minimal()
```



The best appraoch to classify glass type for this dataset is random forests.

The test set errors for classification of `typeclass` is plotted.

```
typeclass_df = data.frame("model" = c("boosted tree", "knn",
                                      "logistic regression"),
                          "error" = c(error_boost, error_knn, error_log))
ggplot(typeclass_df, aes(x = reorder(model, error), y = error)) +
  geom_col(color = "lemonchiffon4", fill = "lemonchiffon1") +
  labs(x = "modeling approach", y = "test set misclassification error") +
  ggtitle("Test Set Error based on Modeling Approach to Predict Typeclass") +
  theme_minimal()
```



The best approach to classify glass type class is the knn classifier, followed by the boosted tree.

All of the lab instructions in this document are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.