Practice1

Umaimah Ahmed

2025-03-06

The overall assignment is due by midnight, Thursday, March 6th to Gradescope. The intermediate part of the assignment - see separate checklist and instructions - is due by midnight, Thursday, Feb. 27th to Gradescope.

Practicing Academic Integrity

If you worked with others or used resources outside of provided course material (anything besides our textbook(s), course materials in Moodle, R help menu) to complete this assignment, please acknowledge them below using a bulleted list. Generative AI is not permitted for our course.

I acknowledge the following individuals with whom I worked on this assignment:

Name(s) and corresponding problem(s)

-

I used the following sources to help complete this assignment:

Source(s) and corresponding problem(s)

-

Prompt

The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence... if it is correctly identified! (Spiehler 1987) It turns out that glass isn't always easily identifiable. In order to help criminologists classify glass, data was gathered in order to allow for analysis and development of appropriate models.

Note that the variable ID is NOT a variable that should be used in the classification. It is just the observation number. It is removed for you below.

Data Description from source:
1. ID number (should not be used in the analysis)
2. RI: refractive index
3. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 3-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron
11. Type of glass: (class attribute)
– 1 building_windows_float_processed
– 2 building_windows_non_float_processed
– 3 vehicle_windows_float_processed
– 4 vehicle_windows_non_float_processed
– 5 containers
– 6 tableware
– 7 headlamps

Your task is to explore the data set and propose at least 3 classification models (from different techniques; at least one tree-based technique and at least one non-tree-based technique must be used, a total of (at least) 3 different techniques must be used) to classify the glass for the criminologists. Then, you must provide a final model with appropriate support for why it is the best model to use.

There is some flexibility with how you *orient* yourself to the problem. For example, if you want to focus on building models that might be easily explainable in court cases for evidence, you might take a different approach than someone who is not concerned about that. You get to discuss your choice/understanding of the problem in the introduction.

Audience: Remember in the methods section that your audience has covered all the methods before classification, so you don't need to re-explain CV, etc. But they need an explanation of the classification methods you are using. The audience is not (necessarily) the criminologists, but a statistically literate audience.

Introduction

The classification of glass types plays an important role in forensic investigations, where correctly identifying glass fragments found at crime scenes can provide valuable evidence. However, distinguishing between different types of glass can prove to be difficult. To assist criminologists in making accurate classifications, this study explores three different classification methods to find an effective and efficient model that will categorize glass samples into one of six types of glass based on key attributes such as refractive index or chemical make-up. We will use two tree-based modeling techniques (random forest and gradient boosting), and one non-tree-based method (neural nets). Our aim is to find a model that balances accuracy and efficiency, aiding criminologists and forensic teams in analyzing future glass samples.

Data

Missing Data and ZV/NZV

The glass data set had 214 observations with 10 variables relevant to our analysis. There were 9 quantitative predictor variables and 1 categorical variable, which was the response we aimed to predict. The predictor variables are as follows:

- `RI`, the refractive index of the glass,
- `Na`, the weight percent of sodium in the glass sample,
- `Mg`, the weight percent of magnesium,
- `Al`, the weight percent of aluminum,
- `Si`, the weight percent of silicon,
- `K`, the weight percent of potassium,
- `Ca`, the weight percent of calcium,
- `Ba`, the weight percent of barium, and
- `Fe`, the weight percent of iron,

The response variable, `Type` had 7 possible levels,

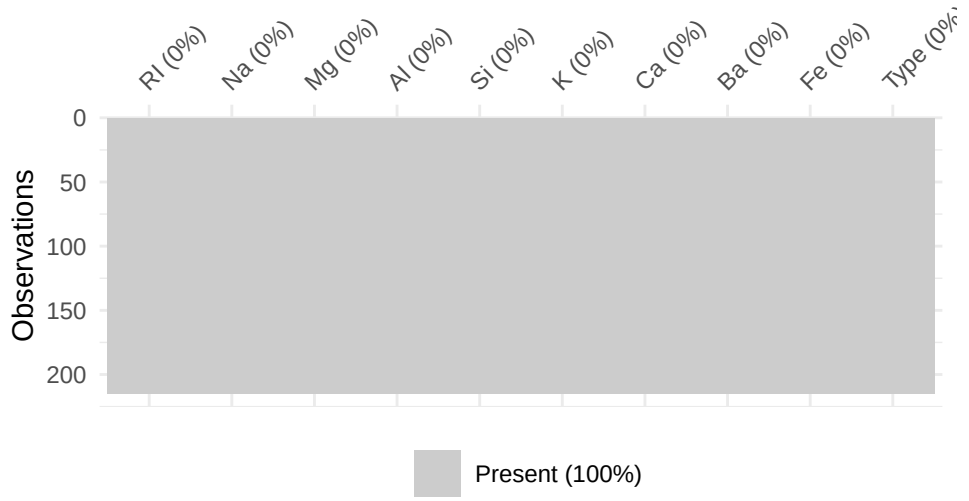
```
-- 1 building_windows_float_processed
-- 2 building_windows_non_float_processed
-- 3 vehicle_windows_float_processed
-- 4 vehicle_windows_non_float_processed
-- 5 containers
-- 6 tableware
-- 7 headlamps
```

We started by processing the data. We made sure the response variable, `Type`, was coded in a way that allows us to do analysis. We also removed the `ID` variable because it's just for identification and unnecessary for the rest of our analysis.

```
glass <- read.table("https://awagaman.people.amherst.edu/stat240/glass.txt",
                    h = T, sep = ",")
glass <- mutate(glass, Type = paste("Type", Type, sep=""))
glass <- mutate(glass, Type = factor(Type)) %>%
  dplyr::select(-ID) #removes ID
```

First we checked the data set to see if there was any missingness or any zero/near-zero variance variables.

```
# visualization of missing data
visdat::vis_miss(glass, cluster = TRUE)
```




```
# checking variance
caret::nearZeroVar(glass, saveMetrics = TRUE) %>%
tibble::rownames_to_column() %>%
filter(nzv)
```

```
## [1] rowname       freqRatio     percentUnique zeroVar       nzv
## <0 rows> (or 0-length row.names)
```

There were no missing values. None of the variables appeared to have zero or near-zero variance.

Exploratory Data Analysis

```
### response variable ###
# distribution of classes
tally(~Type, data = glass, format = "count")
```

```
## Type
## Type1 Type2 Type3 Type5 Type6 Type7
##    70    76    17    13     9    29
```

```
n <- nrow(glass)
(n-76)/n
```

```
## [1] 0.64486
```

```
1-(n-76)/n
```

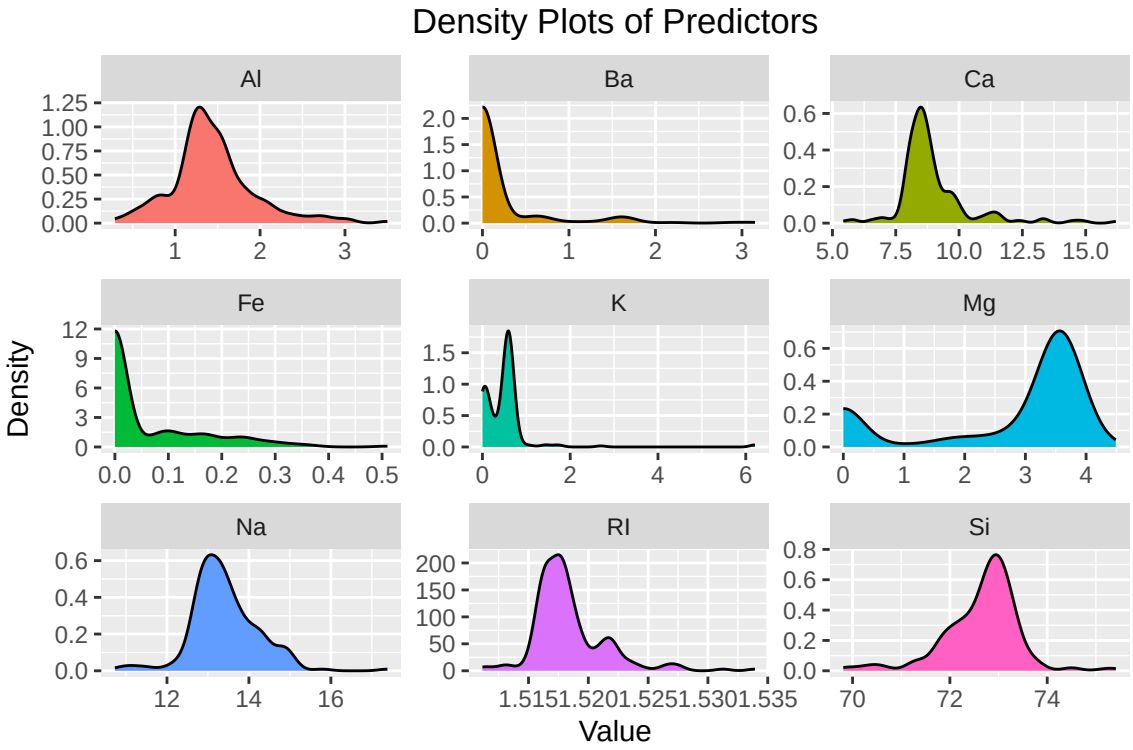
```
## [1] 0.35514
```

There are six types of glass in our analysis: float glass building window (70), non-float glass building window (76), float glass vehicle window (17), non-float glass vehicle window (0), containers (13), tableware (9), headlamps (29). Although non-float glass vehicle window was present as a level for the Type variable, there weren't any observations for that level. It looks like there's some serious class imbalance. We'll have to stratify by Type when we split the data to make sure no class is lost.

The misclassification error rate if we classified every observation as the most prevalent type of glass, non-float building window glass, would be 0.6449. In other words, our baseline AER (apparent error rate) is $1 - 0.6449 = 0.3551$. We aim to develop a model with an AER lower than .3551.

```r
# longer data
glass_long <- glass %>%
  pivot_longer(cols = c(RI, Na, Mg, Al, Si, K, Ca, Ba, Fe),
               names_to = "predictor", values_to = "value")

# Create density plot
ggplot(glass_long, aes(x = value)) +
  geom_density(aes(fill = predictor)) +
  facet_wrap(~ predictor, scales = "free") +
  theme(legend.position = "none",
        plot.title = element_text(hjust = 0.5)) +
  labs(title = "Density Plots of Predictors", x = "Value", y = "Density")
```



Taking a look at the distributions of the predictor variables, many of them do not appear to be normally distributed. Several of them, `K`, `Ba`, and `Fe` appear to be heavily skewed.

Methods

Next, we set up our training and test sets, and our cross validation process. To train and test our models and tune hyperparameters, we split the data set with a 75/25 split. We chose this ratio so that there are a sufficient number of observations in the test set while still giving the model enough data to be trained on. We also made sure to stratify the training and test sets by class to maintain similar distributions for `Type`, our reponse variable. During our exploratory data analysis, we noted that there were some very small classes in the response variable, so another reason to stratify was to make sure we didn't lose any classes in the training or test sets. With a smaller data set, we also decided it would be appropriate to do repeated CV; for the models in this study we used 10-fold cross validation repeated 5 times. Cross validation is necessary for hyperparameter tuning (which we will explain in more detail later), allowing us to tweak our model to find appropriate settings to lower rates of misclassification.

```
# splitting data
set.seed(240)
split <- initial_split(glass, prop = 0.75, strata = "Type")
glass_train <- training(split)
glass_test <- testing(split)
```

```
# CV setup
glass_control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  classProbs = TRUE)
```

We will be applying feature engineering with the `recipes` package in `R`. We will start by making a recipe that applies necessary pre-processing steps based on what the classification method requires. Then, it will be prepped on the training set, estimating the parameters needed to properly apply the recipe blueprint. With the recipe prepped, we can bake it onto the training and test sets.

For the classification methods we applied, it was necessary to create two recipe blueprints. Different methods can require different pre-processing steps, and it is important to prepare the training and test sets for models to be fit on them. For our tree-based methods, random forest and gradient boosting, minimal pre-processing was required. In the Data section we saw that there were no zero or near-zero variance variables, so we are able to set the recipe for the tree methods as-is. For kNN, a method that is sensitive to scale, it was necessary for us to standardize the data before training the kNN model.

```
# recipe used for trees
bp_tree <- recipe(Type ~ ., data = glass_train)

# recipe used for knn
bp_scale <- recipe(Type ~ ., data = glass_train) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes())
```

To compare models and measure performance, we will compare the both the apparent error rate (AER) and estimated true error rate (TER) of each proposed model.The AER, calculated directly from the training data, provides an initial assessment of model accuracy but tends to be overly optimistic due to potential over-fitting. To obtain a more reliable estimate, we compute the TER, which better reflects the model's expected performance on unseen data.

This study tests two tree-based classification models and one non-tree based classification model to find the method that best classifies the crime-scene glass data.The only non-tree method we used in this study was the method of k-nearest neighbors. K-nearest neighbors, or KNN, is a nonparametric method that classifies data based on the data points most similar to it. For each observation, we find a specified k number of observations nearest to it. Although there are several different ways to measure what is "nearest," for this study we used

Euclidean distance. Observations are classified to the majority class of the nearest neighbors. For KNN, standardization is a necessary step in pre-processing the data, as distance measurements are sensitive to scale. The only hyperparameter we have to worry about tuning is k, the number of neighbors. KNN can become computationally inefficient with larger data sets but for glass classification in this study, with a relatively low number of observations and only nine predictors, it took only a few minutes to run. Additionally, KNN is rather intuitive, which may prove beneficial when potentially presenting this model in a court as evidence.

The next classification method we employed was random forest. Random forest is a type of ensemble method that improves on the instability of decision trees by aggregating multiple trees. It also addresses the greediness of trees by limiting the number of predictors that can be used. Random forest involves first bootstrapping B samples from the data set, and a tree is made from each bootstrapped sample. The best split for each step is chosen at random from a subset of m_{try} predictors. The hyperparameters we tuned for the random forest model were m_{try} and the minimum node size, which dictates what the size of the smallest node in each tree can be.

The last model is a gradient boosting model, or GBM. Like random forest, gradient boosting is another technique that works by building an ensemble of decision trees. It improves on each tree iteratively, minimizing error by improving where the previous tree fells short. Each new tree thereby "boosts" performance. Boosting starts with a base learner, which was a decision tree in this study, but boosting is a framework that can be used for any weak learner. The base learners start weak and are trained sequentially by targeting rows of data where the previous tree had the largest errors. The learning rate decides the size of the step the boost will take to improve the previous tree. Although a small learning rate will likely lead to higher accuracy, if its *too* small, then the algorithm becomes computationally cumbersome and will take too long to find the minimum gradient. If the learning rate is too big, then it might jump past the minimum gradient and leave us with a tree that doesn't perform very well. The hyperparameters we tuned for this model were the depth of the individual trees and the minimum node size.

Hyperparameter tuning is an important step in the process of building a strong classification model, altering how the model is trained and how well is generalizes to new data. Tuning allows us to find the best combination of hyperparameters that reduces error and improves predictive power. In order to test several hyperparameter combinations, we create hyperparameter grids to perform grid searches. The grid helps us optimize hyperparameter tuning. For random forest, we tried values of m_{try} 1 through 9 (classifying with only one predictor variable, all the way up to all predictor variables), and a minimum node size of 1, 2, 6, 8. For gradient boosting, we held the number of trees constant at 500, the shrinkage rate constant at 0.05, but tested interaction depths of 8, 10 and 15, and minimum node sizes of 8, 6, and 10. For kNN we tried several values of k from 1 through 40. For each classification method, the hyperparameter combination that resulted in the model with with the highest cross validated accuracy was saved to compare accross all methods.

```
# for random forest
glass_rfgrid <- expand.grid(
  mtry = 2:9,
  splitrule = "gini",
  min.node.size = c(2, 4, 5, 6))

# for gradient boosting
glass_boostgrid <- expand.grid(
  n.trees = 1000,
  shrinkage = 0.05,
  interaction.depth = c(8, 10, 15, 20),
  n.minobsinnode = c(4, 6, 8, 10))

# for knn
glass_knngrid <- expand.grid(
  k = seq(1, 41, by = 4))
```

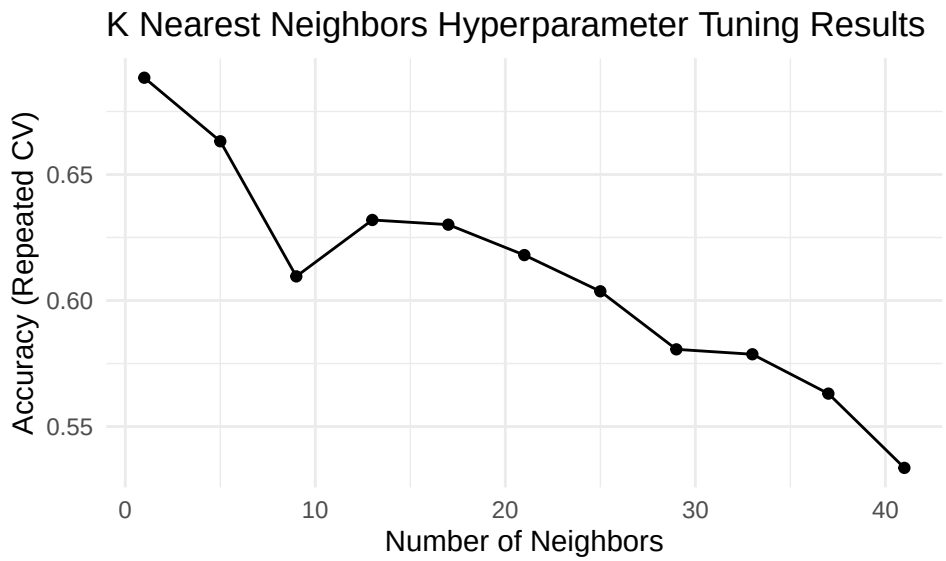
Results

Fitting our Models

We start with a grid search for the k parameter in our KNN model.

```
# fitting knn
set.seed(240)
glass_knntune <- train(
  bp_scale,
  data = glass_train,
  method = "knn",
  trControl = glass_control,
  tuneGrid = glass_knngrid)

ggplot(glass_knntune) +
  labs(title = "K Nearest Neighbors Hyperparameter Tuning Results",
       x = "Number of Neighbors",
       y = "Accuracy (Repeated CV)") +
  theme_minimal()
```




```
glass_knntune
```

```
## k-Nearest Neighbors
##
## 160 samples
##   9 predictor
##   6 classes: 'Type1', 'Type2', 'Type3', 'Type5', 'Type6', 'Type7'
##
## Recipe steps: center, scale
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 143, 144, 145, 143, 144, 143, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy  Kappa
##    1  0.688382  0.570316
##    5  0.663180  0.523951
```

```
##    9  0.609566  0.445494
##   13  0.631948  0.471074
##   17  0.630095  0.462732
##   21  0.618025  0.444429
##   25  0.603656  0.421695
##   29  0.580629  0.387628
##   33  0.578678  0.387077
##   37  0.563067  0.362640
##   41  0.533602  0.312434
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

```
# aer
1-0.688382
```

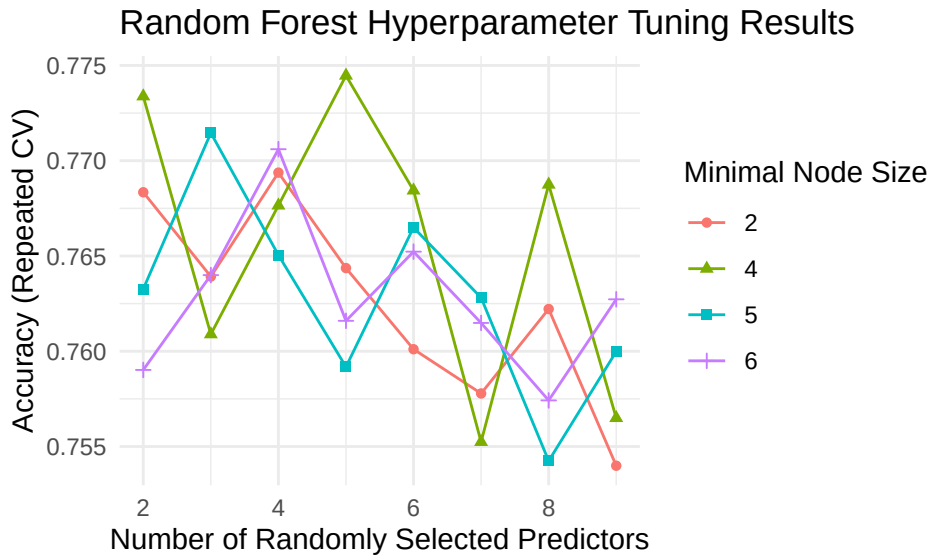
```
## [1] 0.311618
```

The KNN model that used $k = 1$ had the lowest error rate, with an AER of 0.312. While this is better than our benchmark misclassification error of 0.353, it's not much of an improvement.

Next we do a grid search for the random forest model.

```
# fitting random forest
set.seed(240) # for reproducibility
glass_rftune <- train(
  bp_tree,
  data = glass_train,
  method = "ranger",
  trControl = glass_control,
  tuneGrid = glass_rfgrid)

ggplot(glass_rftune) +
  labs(title = "Random Forest Hyperparameter Tuning Results",
       x = "Number of Randomly Selected Predictors",
       y = "Accuracy (Repeated CV)") +
  theme_minimal()
```



```
glass_rftune
```

```
## Random Forest
##
## 160 samples
##   9 predictor
##   6 classes: 'Type1', 'Type2', 'Type3', 'Type5', 'Type6', 'Type7'
##
## Recipe steps:
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 143, 144, 145, 143, 144, 143, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy  Kappa
##   2     2              0.768345  0.675620
##   2     4              0.773378  0.681677
##   2     5              0.763266  0.666695
##   2     6              0.759020  0.660487
##   3     2              0.763920  0.669836
##   3     4              0.760893  0.666022
##   3     5              0.771458  0.681478
##   3     6              0.763995  0.668843
##   4     2              0.769375  0.679189
##   4     4              0.767651  0.675822
##   4     5              0.765049  0.672254
##   4     6              0.770603  0.679757
##   5     2              0.764365  0.674051
##   5     4              0.774468  0.685519
##   5     5              0.759179  0.663929
##   5     6              0.761597  0.667685
##   6     2              0.760107  0.666965
##   6     4              0.768439  0.678259
##   6     5              0.766496  0.675345
##   6     6              0.765226  0.672946
##   7     2              0.757783  0.664435
##   7     4              0.755242  0.659958
##   7     5              0.762810  0.670825
##   7     6              0.761486  0.669120
##   8     2              0.762216  0.671176
##   8     4              0.768746  0.679865
##   8     5              0.754244  0.659069
##   8     6              0.757421  0.663984
##   9     2              0.753992  0.660357
##   9     4              0.756498  0.663360
##   9     5              0.759984  0.668309
##   9     6              0.762726  0.672260
##
## Tuning parameter 'splitrule' was held constant at a value of gini
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 5, splitrule = gini
##  and min.node.size = 4.
```

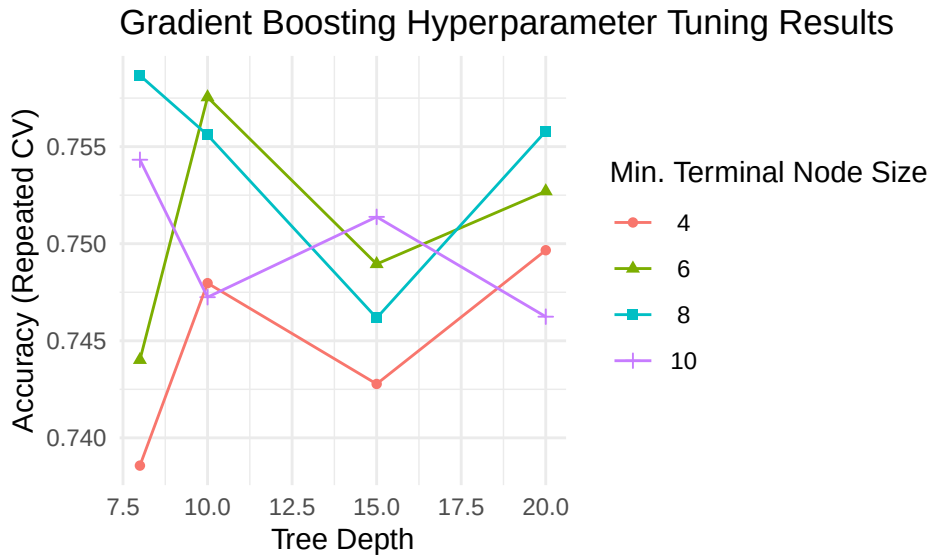
```
# aer
1-0.774468
```

```
## [1] 0.225532
```

It appears that our best random forest model used 5 randomly selected predictors and a minimum node size of 4. The AER for the random forest was 0.226.

```
# gradient boosting
set.seed(240)
glass_boosttune <- train(
  bp_tree,
  data = glass_train,
  method = "gbm",
  trControl = glass_control,
  tuneGrid = glass_boostgrid,
  verbose = FALSE)

ggplot(glass_boosttune) +
  labs(title = "Gradient Boosting Hyperparameter Tuning Results",
       x = "Tree Depth",
       y = "Accuracy (Repeated CV)") +
  theme_minimal()
```




```
glass_boosttune
```

```
## Stochastic Gradient Boosting
##
## 160 samples
##   9 predictor
##   6 classes: 'Type1', 'Type2', 'Type3', 'Type5', 'Type6', 'Type7'
##
## Recipe steps:
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 143, 144, 145, 143, 144, 143, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.minobsinnode  Accuracy  Kappa
##    8                  4               0.738571  0.634238
##    8                  6               0.744016  0.644692
```

```
##    8                  8             0.758666  0.664171
##    8                 10             0.754325  0.656371
##   10                  4             0.747968  0.649380
##   10                  6             0.757545  0.662599
##   10                  8             0.755599  0.659785
##   10                 10             0.747246  0.647895
##   15                  4             0.742775  0.642031
##   15                  6             0.748952  0.651116
##   15                  8             0.746174  0.647403
##   15                 10             0.751383  0.652723
##   20                  4             0.749664  0.649924
##   20                  6             0.752702  0.656793
##   20                  8             0.755791  0.661234
##   20                 10             0.746242  0.645849
##
## Tuning parameter 'n.trees' was held constant at a value of 1000
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.05
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 1000, interaction.depth =
##  8, shrinkage = 0.05 and n.minobsinnode = 8.
```

```
1-0.758666
```

```
## [1] 0.241334
```

With the number of trees and shrinkage rate held constant at 1000 and 0.05 respectively, the best boosting model an interaction depth of 8 and a minimum node size of 8. The cross validated AER was 0.241.

Model Performance

```
# Extract out of sample performance measures
summary(resamples(list(
  model1 = glass_knntune,
  model2 = glass_rftune,
  model3 = glass_boosttune)))
```

```
##
## Call:
## summary.resamples(object = resamples(list(model1 = glass_knntune, model2
##  = glass_rftune, model3 = glass_boosttune)))
##
## Models: model1, model2, model3
## Number of resamples: 50
##
## Accuracy
##            Min.  1st Qu.   Median     Mean  3rd Qu.   Max. NA's
## model1 0.400000 0.614583 0.666667 0.688382 0.764706 0.9375    0
## model2 0.588235 0.705882 0.771242 0.774468 0.848739 1.0000    0
## model3 0.533333 0.705882 0.750000 0.758666 0.820772 0.9375    0
##
## Kappa
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## model1 0.176829 0.467708 0.549508 0.570316 0.678755 0.913514    0
## model2 0.407895 0.583841 0.696699 0.685519 0.788406 1.000000    0
## model3 0.326923 0.595718 0.654014 0.664171 0.755562 0.917098    0
```

If we recall the average cross validated accuracies of each model, it appeared that random forest was performing the best out of the three proposed models. We want to see how the models perform when tested against the training set and compare the AER of each model.

```
### aer
# knn
glass_train_pred_knn <- predict(glass_knntune, glass_train)
confusionMatrix(glass_train_pred_knn, glass_train$Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Type1 Type2 Type3 Type5 Type6 Type7
##      Type1    51     0     0     0     0     0
##      Type2     0    58     0     0     0     0
##      Type3     0     0    12     0     0     0
##      Type5     0     0     0    10     0     0
##      Type6     0     0     0     0     6     0
##      Type7     0     0     0     0     0    23
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.977, 1)
##     No Information Rate : 0.362
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Type1 Class: Type2 Class: Type3 Class: Type5
## Sensitivity                 1.000        1.000        1.000       1.0000
## Specificity                 1.000        1.000        1.000       1.0000
## Pos Pred Value              1.000        1.000        1.000       1.0000
## Neg Pred Value              1.000        1.000        1.000       1.0000
## Prevalence                  0.319        0.362        0.075       0.0625
## Detection Rate              0.319        0.362        0.075       0.0625
## Detection Prevalence        0.319        0.362        0.075       0.0625
## Balanced Accuracy           1.000        1.000        1.000       1.0000
##                      Class: Type6 Class: Type7
## Sensitivity                1.0000        1.000
## Specificity                1.0000        1.000
## Pos Pred Value             1.0000        1.000
## Neg Pred Value             1.0000        1.000
## Prevalence                 0.0375        0.144
## Detection Rate             0.0375        0.144
## Detection Prevalence       0.0375        0.144
## Balanced Accuracy          1.0000        1.000
```

```
# random forest
glass_train_pred_rf <- predict(glass_rftune, glass_train)
confusionMatrix(glass_train_pred_rf, glass_train$Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Type1 Type2 Type3 Type5 Type6 Type7
##      Type1    51     0     0     0     0     0
##      Type2     0    58     0     0     0     0
##      Type3     0     0    12     0     0     0
##      Type5     0     0     0    10     0     0
##      Type6     0     0     0     0     6     0
##      Type7     0     0     0     0     0    23
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.977, 1)
##     No Information Rate : 0.362
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Type1 Class: Type2 Class: Type3 Class: Type5
## Sensitivity                 1.000        1.000        1.000       1.0000
## Specificity                 1.000        1.000        1.000       1.0000
## Pos Pred Value              1.000        1.000        1.000       1.0000
## Neg Pred Value              1.000        1.000        1.000       1.0000
## Prevalence                  0.319        0.362        0.075       0.0625
## Detection Rate              0.319        0.362        0.075       0.0625
## Detection Prevalence        0.319        0.362        0.075       0.0625
## Balanced Accuracy           1.000        1.000        1.000       1.0000
##                      Class: Type6 Class: Type7
## Sensitivity                1.0000        1.000
## Specificity                1.0000        1.000
## Pos Pred Value             1.0000        1.000
## Neg Pred Value             1.0000        1.000
## Prevalence                 0.0375        0.144
## Detection Rate             0.0375        0.144
## Detection Prevalence       0.0375        0.144
## Balanced Accuracy          1.0000        1.000
```

```
# gradient boosting
glass_train_pred_gb <- predict(glass_boosttune, glass_train)
confusionMatrix(glass_train_pred_gb, glass_train$Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Type1 Type2 Type3 Type5 Type6 Type7
##      Type1    51     0     0     0     0     0
##      Type2     0    58     0     0     0     0
##      Type3     0     0    12     0     0     0
##      Type5     0     0     0    10     0     0
```

```
##      Type6     0     0     0     0     6     0
##      Type7     0     0     0     0     0    23
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.977, 1)
##     No Information Rate : 0.362
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Type1 Class: Type2 Class: Type3 Class: Type5
## Sensitivity                 1.000        1.000        1.000       1.0000
## Specificity                 1.000        1.000        1.000       1.0000
## Pos Pred Value              1.000        1.000        1.000       1.0000
## Neg Pred Value              1.000        1.000        1.000       1.0000
## Prevalence                  0.319        0.362        0.075       0.0625
## Detection Rate              0.319        0.362        0.075       0.0625
## Detection Prevalence        0.319        0.362        0.075       0.0625
## Balanced Accuracy           1.000        1.000        1.000       1.0000
##                      Class: Type6 Class: Type7
## Sensitivity                1.0000        1.000
## Specificity                1.0000        1.000
## Pos Pred Value             1.0000        1.000
## Neg Pred Value             1.0000        1.000
## Prevalence                 0.0375        0.144
## Detection Rate             0.0375        0.144
## Detection Prevalence       0.0375        0.144
## Balanced Accuracy          1.0000        1.000
```

Just based on the AER, we can't tell which model performed the best. All three models performed extremely well on the training set. So, we decided to take the a look at the estimated true error rate, the TER. We estimate the true error by testing the model against the test set.

```
### ter
# knn
glass_test_pred_knn <- predict(glass_knntune, glass_test)
confusionMatrix(glass_test_pred_knn, glass_test$Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Type1 Type2 Type3 Type5 Type6 Type7
##      Type1    12     1     2     0     0     0
##      Type2     4    15     2     0     0     0
##      Type3     3     1     1     0     0     0
##      Type5     0     0     0     3     0     0
##      Type6     0     1     0     0     3     0
##      Type7     0     0     0     0     0     6
##
```

```
## Overall Statistics
##
##                Accuracy : 0.741
##                  95% CI : (0.603, 0.85)
##     No Information Rate : 0.352
##     P-Value [Acc > NIR] : 6.55e-09
##
##                   Kappa : 0.652
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Type1 Class: Type2 Class: Type3 Class: Type5
## Sensitivity                 0.632        0.833       0.2000       1.0000
## Specificity                 0.914        0.833       0.9184       1.0000
## Pos Pred Value              0.800        0.714       0.2000       1.0000
## Neg Pred Value              0.821        0.909       0.9184       1.0000
## Prevalence                  0.352        0.333       0.0926       0.0556
## Detection Rate              0.222        0.278       0.0185       0.0556
## Detection Prevalence        0.278        0.389       0.0926       0.0556
## Balanced Accuracy           0.773        0.833       0.5592       1.0000
##                      Class: Type6 Class: Type7
## Sensitivity                1.0000        1.000
## Specificity                0.9804        1.000
## Pos Pred Value             0.7500        1.000
## Neg Pred Value             1.0000        1.000
## Prevalence                 0.0556        0.111
## Detection Rate             0.0556        0.111
## Detection Prevalence       0.0741        0.111
## Balanced Accuracy          0.9902        1.000
```

```
# random forest
glass_test_pred_rf <- predict(glass_rftune, glass_test)
confusionMatrix(glass_test_pred_rf, glass_test$Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Type1 Type2 Type3 Type5 Type6 Type7
##      Type1    16     1     3     0     0     0
##      Type2     3    15     1     1     0     0
##      Type3     0     0     1     0     0     0
##      Type5     0     1     0     2     0     0
##      Type6     0     1     0     0     3     0
##      Type7     0     0     0     0     0     6
##
## Overall Statistics
##
##                Accuracy : 0.796
##                  95% CI : (0.665, 0.894)
##     No Information Rate : 0.352
##     P-Value [Acc > NIR] : 2.92e-11
##
##                   Kappa : 0.719
```

```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Type1 Class: Type2 Class: Type3 Class: Type5
## Sensitivity                 0.842        0.833       0.2000       0.6667
## Specificity                 0.886        0.861       1.0000       0.9804
## Pos Pred Value              0.800        0.750       1.0000       0.6667
## Neg Pred Value              0.912        0.912       0.9245       0.9804
## Prevalence                  0.352        0.333       0.0926       0.0556
## Detection Rate              0.296        0.278       0.0185       0.0370
## Detection Prevalence        0.370        0.370       0.0185       0.0556
## Balanced Accuracy           0.864        0.847       0.6000       0.8235
##                      Class: Type6 Class: Type7
## Sensitivity                1.0000        1.000
## Specificity                0.9804        1.000
## Pos Pred Value             0.7500        1.000
## Neg Pred Value             1.0000        1.000
## Prevalence                 0.0556        0.111
## Detection Rate             0.0556        0.111
## Detection Prevalence       0.0741        0.111
## Balanced Accuracy          0.9902        1.000
```

```r
# boosting
glass_test_pred_gb <- predict(glass_boosttune, glass_test)
confusionMatrix(glass_test_pred_gb, glass_test$Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Type1 Type2 Type3 Type5 Type6 Type7
##      Type1    16     1     3     0     0     0
##      Type2     3    15     0     1     0     0
##      Type3     0     0     2     0     0     0
##      Type5     0     1     0     2     0     0
##      Type6     0     1     0     0     3     0
##      Type7     0     0     0     0     0     6
##
## Overall Statistics
##
##                Accuracy : 0.815
##                  95% CI : (0.686, 0.907)
##     No Information Rate : 0.352
##     P-Value [Acc > NIR] : 3.89e-12
##
##                   Kappa : 0.746
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Type1 Class: Type2 Class: Type3 Class: Type5
## Sensitivity                 0.842        0.833       0.4000       0.6667
## Specificity                 0.886        0.889       1.0000       0.9804
```

```
## Pos Pred Value              0.800        0.789       1.0000       0.6667
## Neg Pred Value              0.912        0.914       0.9423       0.9804
## Prevalence                  0.352        0.333       0.0926       0.0556
## Detection Rate              0.296        0.278       0.0370       0.0370
## Detection Prevalence        0.370        0.352       0.0370       0.0556
## Balanced Accuracy           0.864        0.861       0.7000       0.8235
##                      Class: Type6 Class: Type7
## Sensitivity                1.0000        1.000
## Specificity                0.9804        1.000
## Pos Pred Value             0.7500        1.000
## Neg Pred Value             1.0000        1.000
## Prevalence                 0.0556        0.111
## Detection Rate             0.0556        0.111
## Detection Prevalence       0.0741        0.111
## Balanced Accuracy          0.9902        1.000
```

```
1-0.796
```

```
## [1] 0.204
```

```
1-0.815
```

```
## [1] 0.185
```

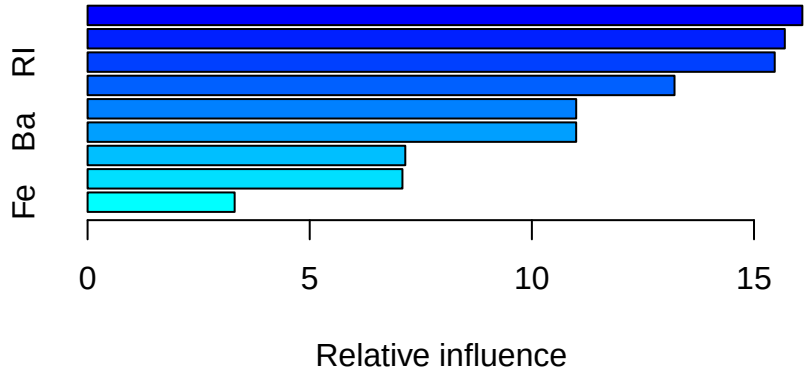
Based on the TER, it appears that random forest has an estimated TER of 0.204% while the GBM had an estimated TER that was slightly better, at 0.185. All of the models seem to be having the most trouble with classifying `Type3` glass, but GBM does a slightly better job than the other two methods.

Next we took a look at the variable importance for the GBM.

```
# boosting
set.seed(240)
gl_boost <- gbm(Type ~ ., data = glass_train,
                  distribution = "multinomial",
                  n.trees = 1000,
                  shrinkage = 0.05,
                  interaction.depth = 8,
                  n.minobsinnode = 8)
```

```
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
```

```
summary(gl_boost)
```




```
##    var  rel.inf
## Mg  Mg 16.08798
## Ca  Ca 15.69332
## RI  RI 15.46600
## Al  Al 13.21034
## K    K 10.99759
## Ba  Ba 10.99725
## Si  Si  7.15072
## Na  Na  7.08682
## Fe  Fe  3.30998
```

Magnesium appears to be the most important variable in the GBM. Calcium and refractive index also appear to be important factors in predicting glass types.

Conclusion

Through our analysis, we found that the best classification model to classify the forensic glass samples was the gradient boosting model. While the the random forest took less time to run compared to the boosting model, it just wasn't as accurate. The GBM performed reasonably well, with a repeated, 10-fold cross validated AER of 0.241 and an estimated TER of 0.186.

Despite its strengths, the GBM still has room for improvement. While it can assist in preliminary investigations, its current accuracy is not sufficient for courtroom evidence. Perhaps with a larger data set, we could refine the model further and improve accuracy. Given the importance of accuracy in forensic classification, any model used in legal settings must meet the highest standards.

Additionally, our analysis highlighted that refractive index, along with magnesium and calcium content, are key factors in predicting glass type. This insight could guide future model development and forensic investigations.