Data 622 - HW 2 - Tree Based Models

William Jasmine

2024-03-29

Setup

The following packages are required to rerun this `.rmd` file:

```r
library(tidyverse)
library(rpart)
library(rpart.plot)
library(knitr)
library(gridExtra)
library(ranger)

seed_num <- 42 # common seed value for randomized operations
```

Part 1 - Introduction

The work presented here shows how decision trees and random forests can be used to solve classification problems. More specifically, we will show how the output of different decision trees affects the quality of the classification predictions, and how those results stack up against a random forest, which aggregates the results of multiple decision trees.

The dataset used here was sourced from Kaggle, and contains information on various mushroom species. Information on how the data was originally collected can be found in this article.

Part 2 - Import/Define the Data

The cell below imports the data from the `mushroom_cleaned.csv` file and stores it as a dataframe, `mush`:

```r
mush <- read.csv("data/mushroom_cleaned.csv")
kable(mush[1:5,], caption='First 5 rows of mush dataframe:')
```

Table 1: First 5 rows of mush dataframe:

cap.diameter	cap.shape	gill.attachment	gill.color	stem.height	stem.width	stem.color	season	class
1372	2	2	10	3.807467	1545	11	1.8042727	1
1461	2	2	10	3.807467	1557	11	1.8042727	1
1371	2	2	10	3.612496	1566	11	1.8042727	1
1261	6	2	10	3.787572	1566	11	1.8042727	1
1305	6	2	10	3.711971	1464	11	0.9431946	1

Each row in the `mush` dataframe represents measurements taken for a single mushroom, with the `class` field in `mush` referring to whether or not the mushroom is edible. This is a binary field with each value representing the following:

- 0 = edible/non-poisonous
- 1 = non-edible/poisonous

The remaining eight fields are all feature variables that quantitatively/qualitatively describe the mushroom, and are defined as follows:

Field Name	Variable Type/Data Type	Description
`cap.diameter`	Continuous/Float	The diameter of the mushroom's cap in m^{-4}.
`cap.shape`	Categorical/Integer	A value from 0-6 the represents the shape of the mushroom cap. These include "bell", "conical", "convex", "flat", "sunken", "spherical", and "others", respectively.
`gill.attachment`	Categorical/Integer	A value from 0-6 the represents how the mushroom gills are connected to its cap. These include "adnate", "adnexed", "decurrent", "free", "sinuate", "pores", and "unknown", respectively.
`gill.color`	Categorical/Integer	A value from 0-11 the represents the color of the mushroom's gills. These include "brown", "buff", "gray", "green", "pink", "purple", "red", "white", "yellow", "blue", "orange", and "black", respectively.
`stem.height`	Continuous/Float	The height of the mushroom's stem (base to cap) in cm.
`stem.width`	Continuous/Float	The width of the mushroom's stem (at it's base) in $^{-4}$.
`stem.color`	Categorical/Integer	A value from 0-11 the represents the color of the mushroom's cap. See `gill.color` for list of colors.
`season`	Continuous/Float	A value that represents the time of year the measurements for the mushroom were taken. Larger values represent measurements that were taken further along in the year.

Based on the below, the `mush` dataframe contains measurements of 53,035 mushrooms:

```
nrow(mush)
```

```
## [1] 54035
```

Of these observations, the following output summarizes the quantity of each that were classified as edible/non-edible:

```
table(mush$class)
```

```
##
##     0     1
## 24360 29675
```

Part 3 - Clean Data

As a first data cleaning step, the cell below checks for any missing/NA values in `mush`:

```
sum(is.na(mush))
```

```
## [1] 0
```

As we can see from the above, this dataset has the benefit of being complete: no data removal/imputation methodologies are necessary.

Based off the field descriptions, the continuous fields have different units of measurement. As such, the field below converts all these fields to be in units of mm (m^{-3}):

```
mush$cap.diameter <- mush$cap.diameter / 10
mush$stem.height <- mush$stem.height * 10
mush$stem.width <- mush$stem.width / 10
```

Based off the output in the previous section, we know that `mush` is currently a imbalanced dataset in which more observations are classified as being non-edible. To simplify making predictions on `class` the following cell undersamples the data via random selection to ensure the classes are balanced:

```
set.seed(seed_num)

class_0 <- which(mush$class == 0)
class_1 <- which(mush$class == 1)
class_1 <- sample(class_1, size=length(class_0), replace=FALSE)
selected_rows <- c(class_0, class_1)
mush <- mush[selected_rows,]

table(mush$class)
```

```
##
##     0     1
## 24360 24360
```

Finally, the cell below converts the `class` field of `mush` to be a factor, which is required by many of the machine learning packages that will be used later in this analysis:

```
mush$class <- as.factor(mush$class)
```

Given the initial cleanliness of the dataset, there is not much else that needs to be cleaned. Note that one-hot-encoding of the categorical features was considered as an additional data cleaning step, but this kind of encoding is not particularly useful for decision tree based classification methods. Furthermore, one-hot-encoding would create an additional 30 feature variables in this case and result in more complex and less interpretible decision trees.

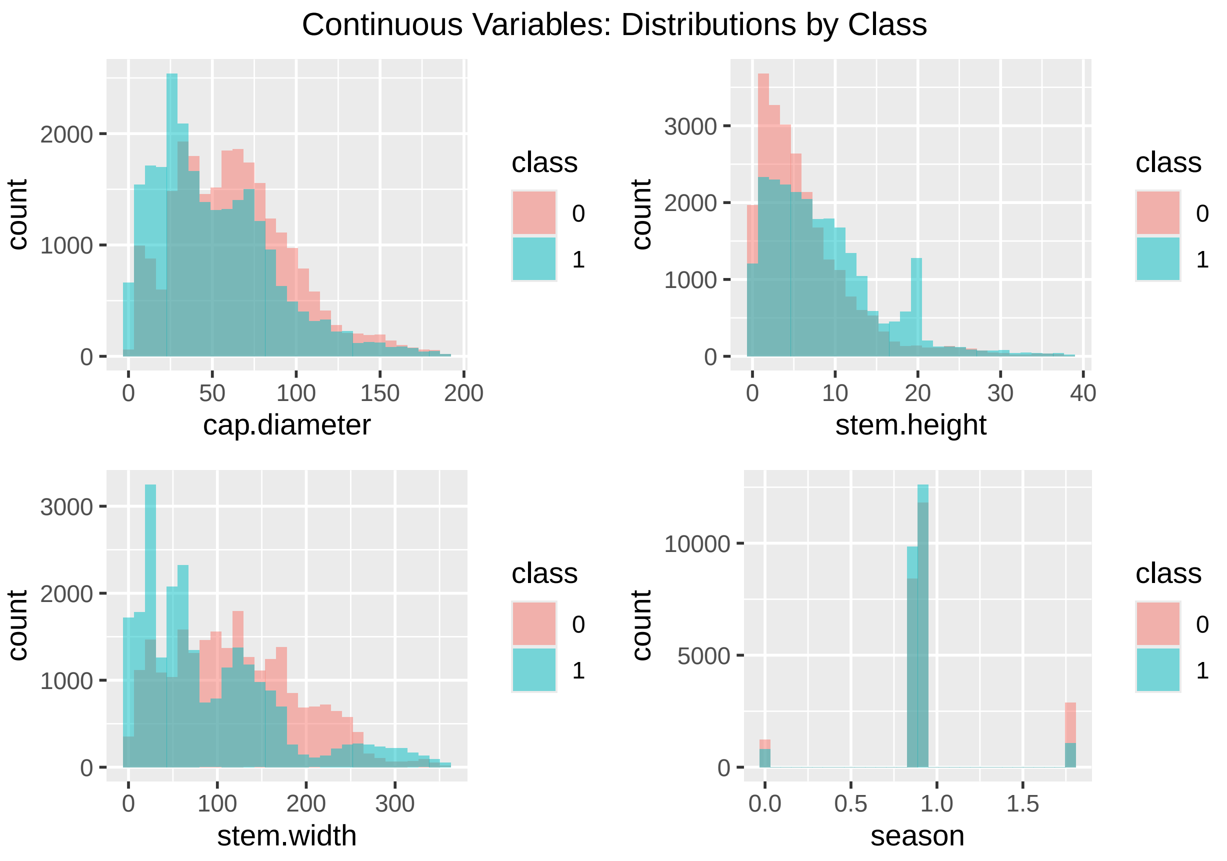
Part 4 - Exploratory Data Analysis (EDA)

To simplify the analysis shown below, the cell below creates new, separate data frames for the categorical and continuous features in `mush`:

```r
features <- mush %>%
  select(-class)

field_types <- unlist(lapply(features, is.integer), use.names = FALSE)
cat_df <- features[which(field_types)]
cont_df <- features[-which(field_types)]
cat_names <- colnames(cat_df)
cont_names <- colnames(cont_df)
```

The following provides histograms of the continuous fields in `mush`, colored by `class`:

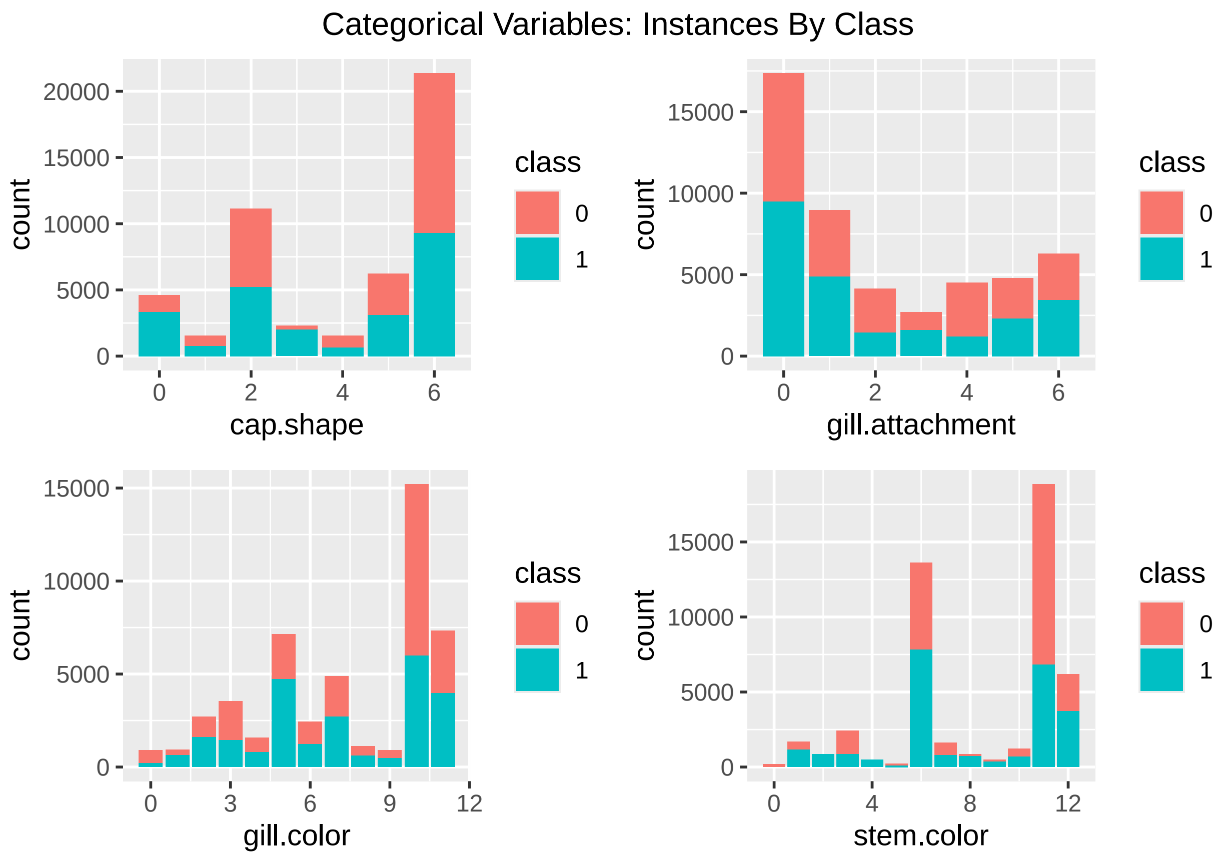
```r
plot_list <- list()

i = 1
for(field_name in cont_names){
  plot_list[[i]] <-
    ggplot(data=mush, aes_string(x=field_name, fill="class")) +
      geom_histogram(alpha=0.5, position = "identity" )
  i = i + 1
}

grid.arrange(grobs=plot_list, ncol=2,
             top='Continuous Variables: Distributions by Class')
```



We can see that there are indeed some differences in the distributions by class for some of the continuous variables, which is an indicator that they might be useful when building the tree based models.

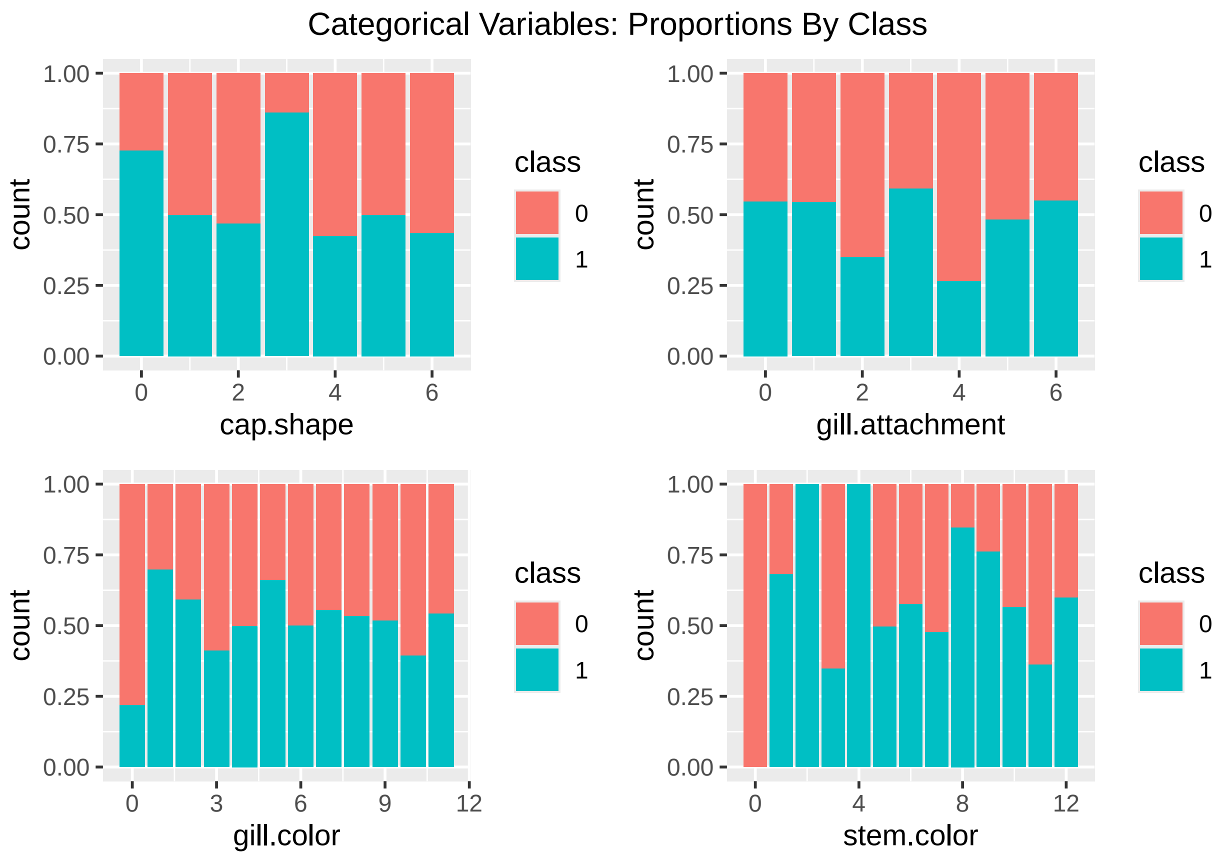
Next, the cell below creates bar graphs for all the categorical variables, once again separated by class:

```r
plot_list <- list()

i = 1
for(field_name in cat_names){
  plot_list[[i]] <-
    ggplot(data=mush, aes_string(x=field_name, fill="class")) +
      geom_bar()
  i = i + 1
}

grid.arrange(grobs=plot_list, ncol=2,
             top ='Categorical Variables: Instances By Class')
```



Most of the different categories for each feature seem to have a relatively equal distribution by class, but this can be further scrutinized by instead looking at the proportions of each class:

```r
plot_list <- list()

i = 1
for(field_name in cat_names){
  plot_list[[i]] <-
    ggplot(data=mush, aes_string(x=field_name, fill="class")) +
      geom_bar(position="fill")
  i = i + 1
}
grid.arrange(grobs=plot_list, ncol=2,
             top='Categorical Variables: Proportions By Class')
```



This plots shown above are more useful, in that they make clear that there are a number of categories in these fields that skew to a single class. For example, 75% of observations with `cap.shape`= 3 are classified as being poisonous. Furthermore, some of the categories in `stem.color` all belong to a single class, but if we compare those categories to the previous plots we see that there are not many instances of observations falling into these categories. Despite this, they will no doubt be useful in our decision tree models.

Part 5 - Decision Trees

The following section shows the result of making predictions using three different decision trees: one using the categorical fields, one using the continuous fields, and one using all explanatory fields. Note that this separation is arbitrary, and is simply for the purpose of analyzing the differences between them.

However, before building the tree models, we need to split the data into training and testing data sets. This is done in the cell below:

```
set.seed(seed_num)

# define training size
train_size <- 0.75

# create separate dataframes by class
class_0_df <- mush[which(mush$class == 0),]
```

```
class_1_df <- mush[which(mush$class == 1),]

# determine row numbers of samples
n <- floor(nrow(class_0_df) * train_size)
sample_vec <- 1:nrow(class_0_df)
train_samples <- sample(sample_vec, size=n, replace=FALSE)

# sample the class dataframes
train_0_df <- class_0_df[train_samples,]
test_0_df <- class_0_df[-train_samples,]

train_1_df <- class_1_df[train_samples,]
test_1_df <- class_1_df[-train_samples,]

# recombine dataframes
train <- rbind(train_0_df, train_1_df)
test <- rbind(test_0_df, test_1_df)

# shuffle rows
train <- train[sample(1:nrow(train)), ]
test <- test[sample(1:nrow(test)), ]

# split into x and y
train_x <- select(train, -class)
train_y <- select(train, class)

test_x <- select(test, -class)
test_y <- select(test, class)
```

The above leaves us with a testing set, `test`, and a training set `train`.

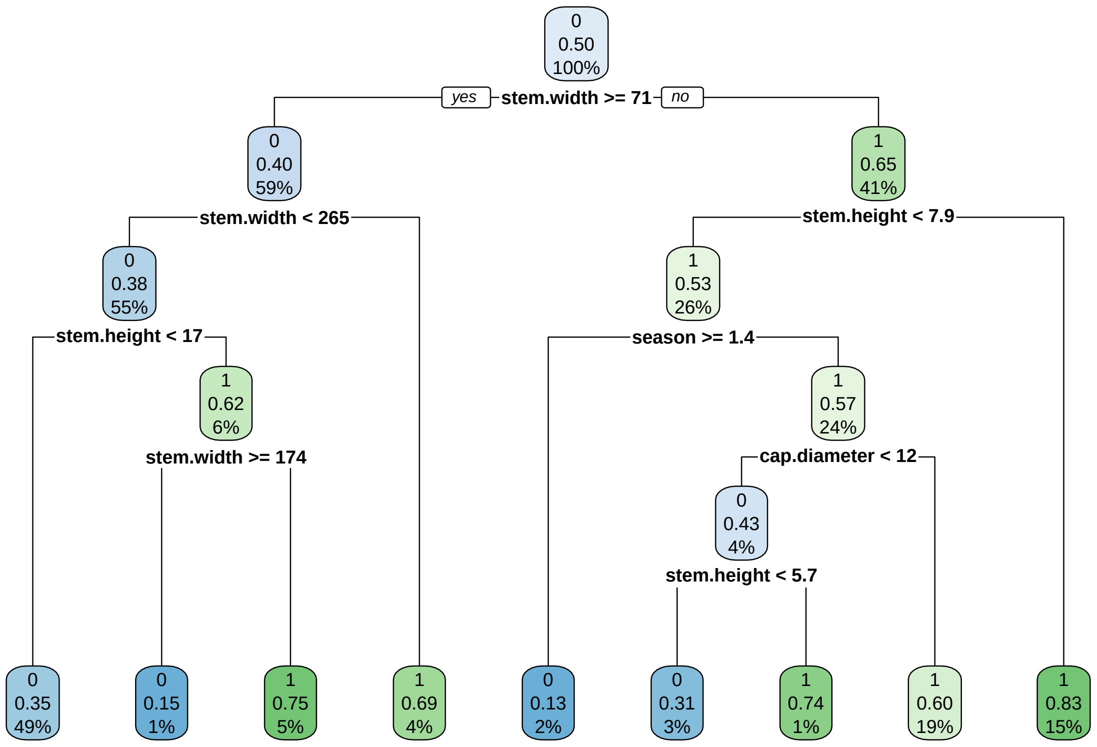
Part 5.1 - Using Continuous Fields

The following cell creates a decision tree using the continuous fields, `stem.height`, `stem.width`, `season`, and `cap.diameter`:

```
set.seed(seed_num)

tree_data <- train[c(cont_names, "class")]
cont_tree <- rpart(class~.,data=tree_data)
```

Now that the model has been created we can use the `rpart.plot` package to make a visualization of our decision tree.

```
rpart.plot(cont_tree)
```



Already, we can make some interesting observations. In the previously made histograms of the continuous features, it is clear that the `stem.width` field has the most pronounced difference in the distributions by class: the class 0 distribution is skewed to the right compared to the class 1 distribution, meaning that class 0 observations tend to have larger values of `stem.width`. In the plot above, we see that `stem.width` is indeed used as the first split in our decision tree, with the majority of observations having a `stem.width` value >= 72 and categorized as class 0. We also see that almost all of the class 0 observations result from each condition being true as we move down the tree (with the leftmost leaf having 49% of all total observations).

The cell below evaluates how the tree performs when making predictions on the test set.

```
cont_preds <- unname(predict(cont_tree, test_x[c(cont_names)], type="class"))
pred_tab <- table(test_y$class, cont_preds)
pred_tab
```

```
##    cont_preds
##        0    1
##   0 4521 1569
##   1 2292 3798
```

In the above confusion matrix we see that the the decision tree correctly classified 4,521 out of 6,090 of the edible mushrooms (~74%), and 3,798 of the 6,090 poisonous mushrooms (~62%). The total accuracy score across both classes is calculated below:

```
cont_score <- sum(diag(pred_tab)) / length(cont_preds)
cont_score
```

```
## [1] 0.6830049
```

Overall, this single decision tree produced respectable result and performed a good deal better than chance. The decision tree did have more of a struggle classifying the poisonous mushrooms, but this is not unexpected given the tree's structure: by summing the percentages in the final row of the tree diagram by class, we see that the model grouped 55% of the training data samples as edible despite the classes being balanced. As such, given that the data in the testing and training set is similar (which it should be given the random sampling), the decision tree is more likely to predict observations as being in the edible class.

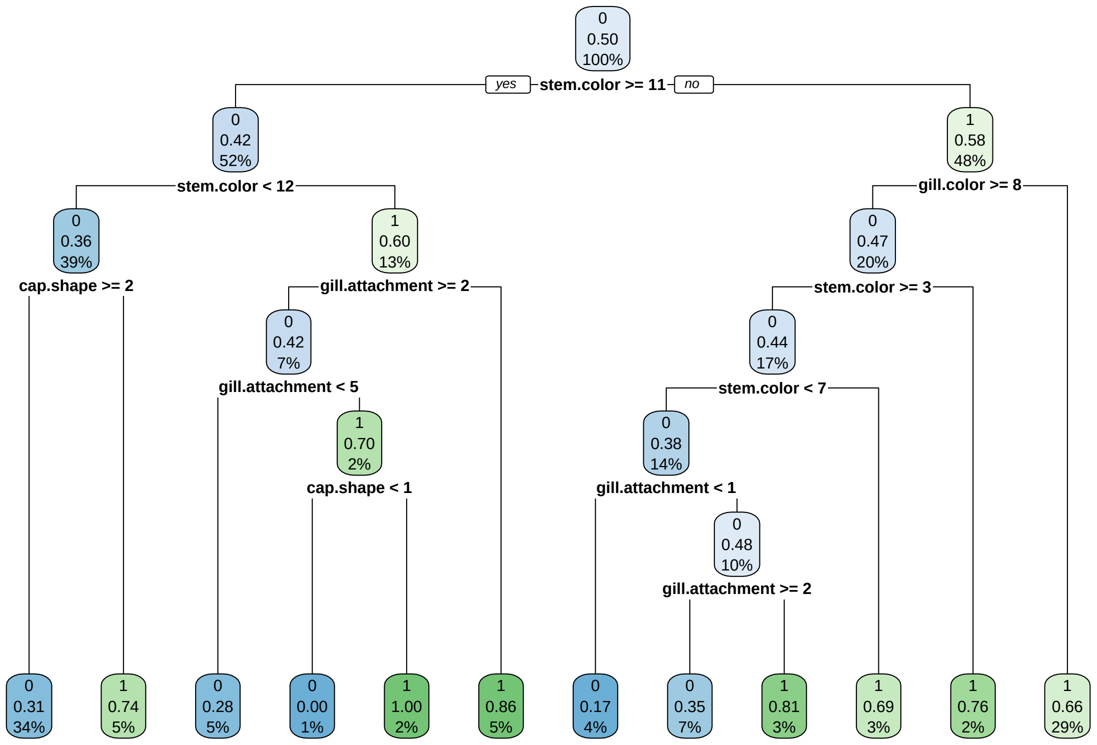
Part 5.2 - Using Categorical Fields

To see if we can make a different decision tree that yields better results, the following cell creates a decision tree using the categorical fields, `gill.attachment`, `gill.color`, `stem.color`, and `cap.shape`:

```
tree_data <- train[c(cat_names, "class")]
cat_tree <- rpart(class~.,data=tree_data)
```

Once again, we build a tree diagram:

```
rpart.plot(cat_tree)
```



and make new predictions:

```
cat_preds <- unname(predict(cat_tree, test_x[c(cat_names)], type="class"))
pred_tab <- table(test_y$class, cat_preds)
pred_tab
```

```
##    cat_preds
##        0    1
##   0 4376 1714
##   1 1839 4251
```

This time around, the decision tree correctly predicted 4376 of the 6,090 edible mushrooms (~72%) and 4,251 of the 6,090 poisonous mushrooms (~70%) resulting in a total accuracy score of:

```
cat_score <- sum(diag(pred_tab)) / length(cat_preds)
cat_score
```

```
## [1] 0.7082923
```

In this case, we ended up with a more complex model (12 final leaves as opposed to 9 in the continuous fields decision tree), which appears to have produced slightly more successful results. In addition, we see less of a difference in performance between classes. The improved accuracy score on the test set compared to the first tree is an indicator of this decision tree having a lower variance.

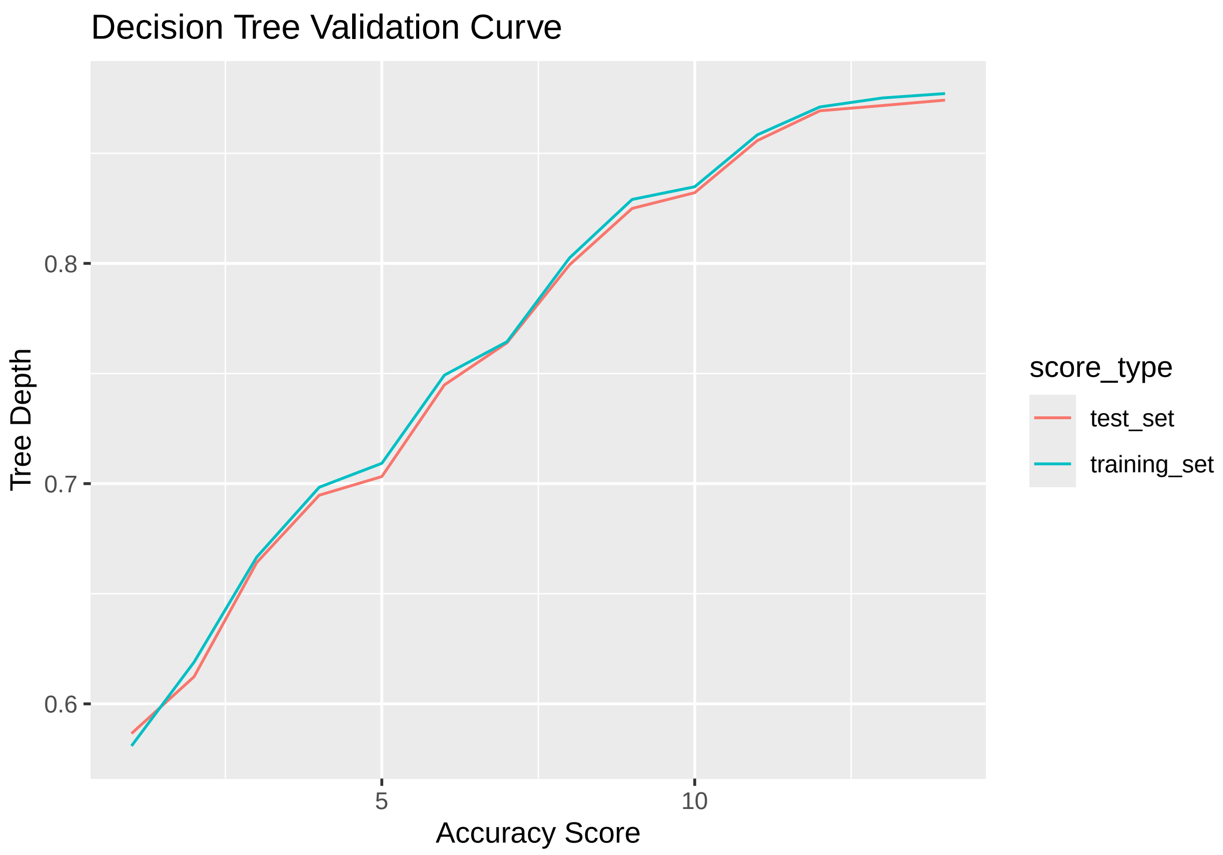
Part 5.3 - Model Complexity Impact on Bias and Variance

To further evaluate how the complexity of the model impacts its ability to make predictions, the cell below uses the categorical fields to create decision trees of different depths and evaluate their performance on both the training and testing sets.

```r
tree_data <- train[c(cat_names, "class")]

i = 1
score_mat <- matrix(ncol = 3, nrow=0)

while(TRUE){
  tree <-
    rpart(class~.,data=tree_data, control=rpart.control(maxdepth = i, cp=-1))

  test_preds <- unname(predict(tree, test_x[c(cat_names)], type="class"))
  train_preds <- unname(predict(tree, train_x[c(cat_names)], type="class"))

  test_pred_tab <- table(test_y$class, test_preds)
  test_score <- sum(diag(test_pred_tab)) / length(test_preds)

  if(test_score == score_mat[i-1,3] && i > 1) break;

  train_pred_tab <- table(train_y$class, train_preds)
  train_score <- sum(diag(train_pred_tab)) / length(train_preds)

  scores <- c(i, train_score, test_score)
  score_mat <- rbind(score_mat, scores)
  i = i + 1
}

score_df <- as.data.frame(score_mat)
colnames(score_df) <- c('depth', 'training_set', 'test_set')
rownames(score_df) <- NULL
```

The loop in the above code block terminates once the `rpart` function that generates the decision trees refuses to make a more complex model. The cell below plots the results of these calculations:

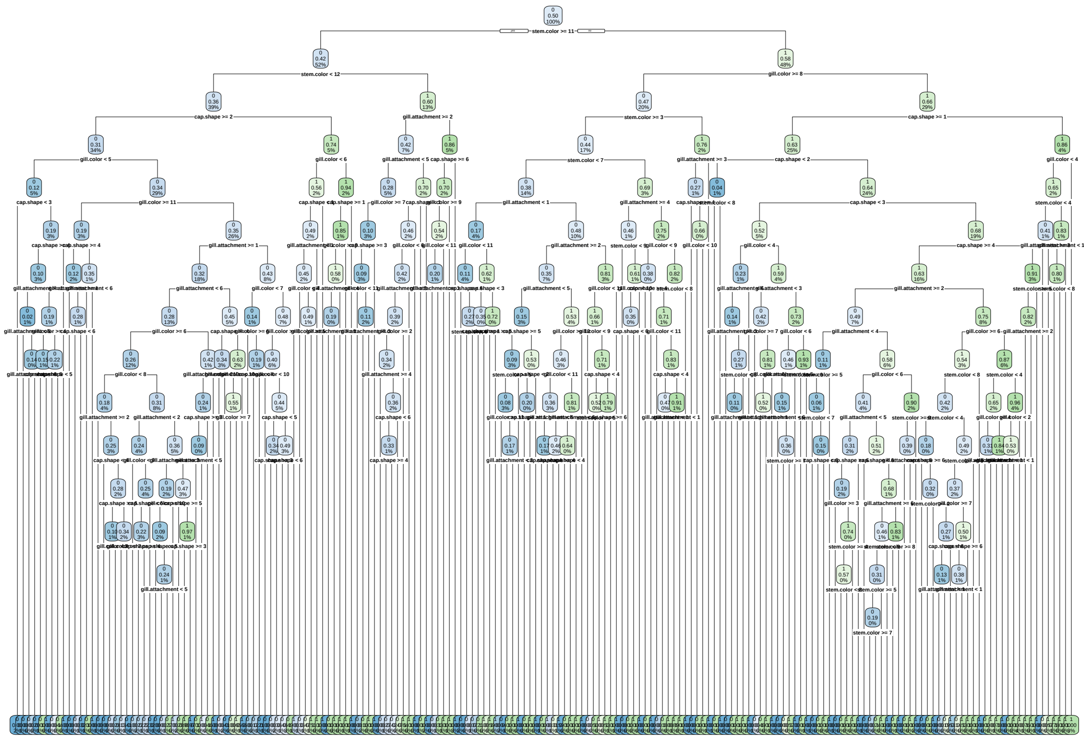
```r
plt_data <-
  pivot_longer(score_df,
               cols=ends_with("set"),
               names_to="score_type",
               values_to = "score")

ggplot(data=plt_data, aes(x=depth, y=score, color=score_type)) + geom_line() +
  labs(x='Accuracy Score',
       title="Decision Tree Validation Curve",
       y='Tree Depth',
       legend='Dataset')
```



Not surprisingly, we see that the training set performance increases with model complexity, which leads to a more overfitted decision tree that will likely have high bias. However, we also see accuracy score continuing to rise for test set accuracy (with these scores bring only slightly lower than the training data at each depth). Given that the model has not yet seen the test data, we would instead expect that its predictions would eventually decrease with model complexity resulting in higher variance models.

This unexpected behavior might be due to the fact that the test data too closely resembles the training data (maybe too many observations of the same species of mushrooms), but it seems unlikely that when new outside data is presented to these high depth models that they would result with the same high levels of accuracy. Below shows a diagram of the most complex model produced:

```
rpart.plot(tree)
```



In this case, the model is much too complex to quickly draw out any helpful findings: there are a total of 184 different outcomes in this case, which will almost definitely will result in overfitting.

```r
sum(tree$frame$var == "<leaf>")
```

```
## [1] 184
```

Part 5.4 - Using all Features

Lastly for this section, the code below produces a decision tree using all of the features, and calculates the accuracy score on the predictions of the test set:

```r
tree <- rpart(class~., data=train)
preds <- unname(predict(tree, test_x, type="class"))
pred_tab <- table(test_y$class, preds)
score <- sum(diag(pred_tab)) / length(preds)
score
```

```
## [1] 0.7384236
```

We see that this decision tree performed the best of them all, with a total accuracy score of approximately 74%. The trade off of using this decision tree is that it requires increased data processing, and results in a more complex model.

Part 6 - Random Forest Model

Random forest classifiers aggregate the results of numerous decision trees, which generally result in models that have increased predictive power. The cell below uses the `ranger` package to produce a random forest using all of the feature variables:

```
rf <- ranger(class~., data=train, probability = TRUE, importance = 'impurity')
print(rf)
```

```
## Ranger result
##
## Call:
##  ranger(class ~ ., data = train, probability = TRUE, importance = "impurity")
##
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      36540
## Number of independent variables:  8
## Mtry:                             2
## Target node size:                 10
## Variable importance mode:         impurity
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.01306345
```

The accuracy score of this random forest model is computed below:

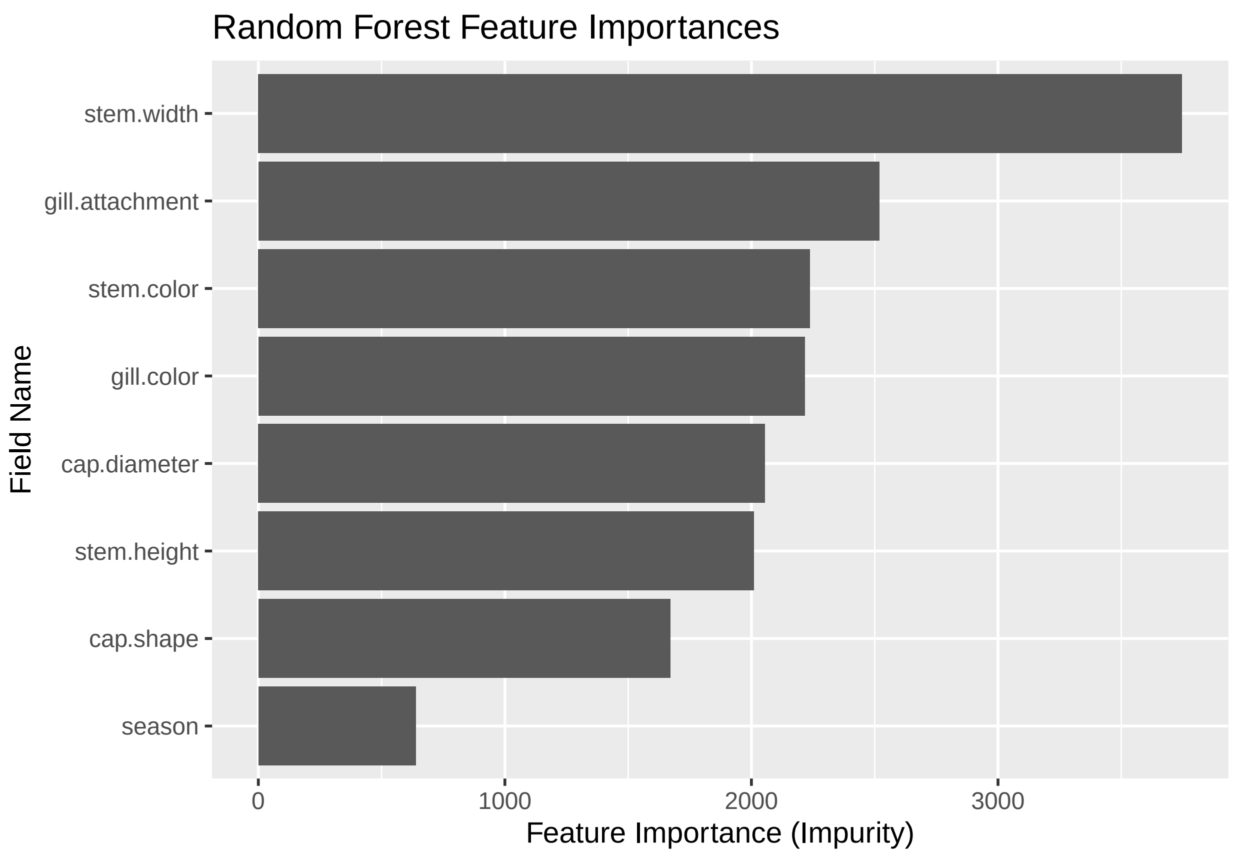
```
preds <- as.data.frame(predict(rf, data=test)$predictions)
colnames(preds) <- c('edible', 'poisonous')
preds <- preds %>%
  mutate(pred = ifelse(edible>.5, 0, 1))
preds <- as.factor(preds$pred)
pred_tab <- table(test_y$class, preds)
score <- sum(diag(pred_tab)) / length(preds)
score
```

```
## [1] 0.9885878
```

As we can see from the output above, the “out-of-the-box” random forest classifier does extremely well in predicting the outcomes of the test set, far exceeding that of the single decision trees used previously. There is no doubt this is due to the aforementioned aggregating of the single decision trees implemented by this model (in this instance, the random forest used 500 different trees).

Random forest models also have the benefit of allowing us to determine the “feature importance” of each explanatory variable - basically a measure of how useful each feature is in the model. The cell below determines these importances and plots them:

```
importances <- as.data.frame(rf$variable.importance)
importances <- cbind(feature = rownames(importances), importances)
rownames(importances) <- 1:nrow(importances)
colnames(importances) <- c('feature', 'importance')

ggplot(importances,
       aes(x=reorder(feature, importance), y=importance)) +
  geom_bar(stat='identity') +
  coord_flip() +
  labs(x='Field Name',
       title="Random Forest Feature Importances",
       y='Feature Importance (Impurity)')
```



In the above graph larger y-values indicate increased importance in the model. We see that the `stem.width` field was the most useful, which makes sense given that we previously identified this field as visually having the most significant difference in distribution by class (referring back to section 4).

Part 7 - Conclusion

Decision trees come with a number of potential drawbacks. This article mentions a number of these, including that they can be hard to create, may become increasingly complex over time, and can be hard to interpret. However, the results shown here prove that even relatively simple decision trees with low depth can provide decent results. Furthermore, packages such as `rpart` make it exceedingly simple to build decision trees that determine the best rules on their own, and they can do this without requiring any prior business context. In fact, instead of analyzing and producing decision trees by hand (something mentioned in the article), these packages allow you to instead produce and then analyze, which will likely generate key insights that are more accurate/helpful.

Furthermore, the results shown here give evidence for more complex models tree-based models - such as random forests - being the clear choice when attempting to make accurate predictions. Though they are more complex and sacrifice a decent level of interpretability, using these models seems like an easy choice when they result in the same level of increased predictive power shown here.

All this being said, as with any machine learning model, the work presented here shows that changing complexity/structure of tree based models has an impact on that model's bias and variance. As such, it is the job of the person(s) producing these models to strike the correct balance of each.

In conclusion, tree based models can be a great choice when presented with classification (or even regression) problems, and their flexibility ensures that there is more likely than not a version of one that meets most applicable real-world scenarios.