Predicting the Edibility of Mushrooms

Delaney Helgeson

STAT 6302 Final Project

Southern Methodist University

May 6th, 2022

Introduction

There are thousands of species of mushrooms that come in a wide variety of colors, shapes and sizes. Whether classifying mushrooms as a hobby or for research purposes, it is critical to be able to distinguish between poisonous and edible mushrooms. The primary goal of this analysis is to detail the process of building and choosing a model to predict the probability that a mushroom is poisonous. Additional research questions include: What factors are important or useful in predicting whether a mushroom is poisonous or edible? Which methods perform better? How could a model like this be used in the context of mycology?

The data set is the Secondary Mushroom Dataset obtained from the UCI machine learning repository[1]. The data set contains 61,069 observations and 21 attributes. There are 17 categorical attributes and 3 quantitative attributes. Mushroom attributes are primarily grouped into four structural categories: the cap, gills, stem, and veil. For each of these four categories, there are variables related to the size, color, and shape. Other attributes include whether or not the mushroom bleeds or bruises, characteristics related to the ring of the mushroom, spore print color, habitat, and season of the mushroom.

Statistical Methods

Data Pre-Processing

In order to prepare the data for model building, it must be pre-processed. There were nine variables that contained missing values; these variables were: cap surface, gill attachment, gill spacing, stem root, stem surface, veil type, veil color, ring type, and spore print color. Table 1 details the percentage of missing values for each of these attributes.

Table 1: Percentage of Missing Values for Incomplete Columns

Variable	**Percent Missing**
Cap surface	23.12
Gill attachment	16.18
Gill spacing	41.04
Stem root	84.39
Stem surface	62.43
Veil type	94.80

[1] https://archive.ics.uci.edu/ml/datasets/Secondary+Mushroom+Dataset

Veil color	87.86
Ring type	4.05
Spore print color	89.60

Veil type and spore print color were removed from the data set, since veil type had over 90% missing values (94.80%) and spore print color had nearly 90% missing values (89.60%). Furthermore, ring type was also removed due to having near zero variance. The criteria for near zero variance was met if (1) the ratio of the frequency of the most prevalent value to the second most prevalent value was greater than 95:5 and (2) the ratio of unique values to the sample size was smaller than 10%. In the case of the ring type variable, the frequency ratio was approximately 99:5, and the unique value ratio was 0.013%, hence its removal.

The data were split uniquely into train, validation, and test sets for model building. The data were split randomly into a train and test set using an 80:20 ratio, and then the train data was randomly subdivided further into a train and validation set using an 80:20 ratio. Ultimately, there were 39,085 observations in the training data, 9,771 observations in the validation data, and 12,213 observations in the test data.

k-Nearest Neighbor imputation was performed on the remaining six incomplete columns: cap surface, gill attachment, gill spacing, stem root, stem surface, and veil color. All other variables were used in calculating the distances from the observation with the missing value. Additionally, all variables were weighted equally in determining the k-nearest neighbor, and all k-nearest neighbors were weighted equally in the aggregation step. The default value of k = 5 was used. Since all incomplete columns were categorical, the mode of the five nearest neighbors was taken as the donor value. If there was a tie for the mode, one level from the tied levels was randomly selected. Imputation was performed separately on the train, validation, and test sets in order to avoid data leakage.

Analysis

The three numeric columns: cap diameter, stem height, and stem width were each fit to a natural cubic spline with 3 degrees of freedom, or 2 knots. Since only three quantitative columns were present in the data, it was desirable to allow for greater flexibility in the relationship to the outcome. The natural cubic splines were calculated separately for the train, validation, and test sets in order to avoid data leakage.

Two models were fit for comparison: a penalized logistic regression and a random forest model. The logistic regression was tuned via 10-fold cross validation repeated 5 times for the strength of parameter penalization (λ) as well as the elastic net penalty (α). Ten equally spaced values between 0.01 and 0.2 were assessed for λ and three values (0, 0.5, and 1) were assessed for α, giving a total of 30 parameter combinations. The three alpha values correspond to ridge, elastic net, and lasso regression, respectively. Accuracy was the metric used to choose the optimal parameters.

Similarly, the random forest model was tuned via 10-fold cross validation repeated 5 times for the number of randomly sampled predictors considered at each split, the impurity metric, and the minimum number of observations in the terminal node. Three values for the number of randomly sampled predictors were assessed: 3, 5, and 7; two impurity metrics were assessed: Gini and Entropy; and three values for the minimum node size were assessed: 1, 3, and 5. The number of bootstrap samples for each random forest was 500. Accuracy was used during training to choose the optimal combination of parameters.

The cutpoint for the dichotomization of probabilities into a class outcome was tuned on the validation data for the logistic regression only. Since the primary objective was to predict the probability that a given mushroom is poisonous, it was desirable to inflate the sensitivity of the model. Afterall, it would be dangerous to falsely identify a poisonous mushroom as edible. Ten potential minimum sensitivity values were chosen, and the cutpoint to achieve each value was identified. The accuracy, Kappa statistic, positive predictive value, negative predictive value, and specificity were calculated on the validation results assuming each of the ten potential cutpoints. The optimal cutpoint was chosen by subjectively considering the trade-off between sensitivity and the other model performance metrics.

Lastly, the final models for both logistic regression and random forest were evaluated on the test data.

Results

Table 2 shows a characteristic summary of the distributions of categorical variables, including the class outcome. Results were tabulated on the training, validation, and test sets combined following pre-processing and imputation.

Table 2: Distributional Summary of Categorical Variables across all Data Sets

Variable	Count	%	Variable	Count	%	Variable	Count	%
Cap shape			Season			Meadows	2920	4.78
Bell	5694	9.32	Spring	2727	4.47	Paths	360	0.59
Conical	1815	2.97	Summer	22898	37.50	Heaths	2001	3.28
Convex	26934	44.10	Autumn	30177	49.41	Urban	115	0.19
Flat	12404	21.95	Winter	5267	8.62	Waste	353	0.58
Sunken	7164	11.73	Gill attachment			Woods	44209	72.39
Spherical	2598	4.25	Adnate	14823	24.27	Stem surface		
Others	3460	5.67	Adnexed	10108	16.55	Fibrous	11021	18.05
Cap surface			Decurrent	11002	18.02	Grooves	4312	7.06
Fibrous	2733	4.48	Free	7792	12.76	Scaly	11679	19.12
Grooves	6004	9.83	Sinuate	6671	10.92	Smooth	17337	28.39
Scaly	7927	12.98	Pores	6785	11.11	Shiny	2128	3.48
Smooth	10040	16.44	None	3888	6.37	Silky	3353	5.49
Shiny	5735	9.39	Gill spacing			Sticky	10171	16.65
Leathery	2922	4.78	Close	42187	69.08	None	1066	1.75
Silky	2880	4.72	Distant	14947	24.28	Stem color		
Sticky	10054	16.46	None	3935	6.44	Brown	18063	29.58
Wrinkled	33302	5.41	Gill color			Buff	173	0.28
Fleshy	4077	6.68	Brown	9645	15.79	Gray	2626	4.30
Dimpled	5395	8.83	Buff	954	1.56	Green	542	0.89
Cap color			Gray	4118	6.74	Pink	1025	1.68
Brown	24218	39.66	Green	1399	2.29	Purple	1490	2.44
Buff	1230	2.01	Pink	5983	9.80	Red	2050	3.36
Gray	4420	7.24	Purple	1023	1.68	White	22926	37.54
Green	1782	2.92	Red	1066	1.75	Yellow	7865	12.88
Pink	1703	2.79	White	18521	30.33	Blue	226	0.37
Purple	1709	2.80	Yellow	9546	15.63	Orange	2187	3.58
Red	4035	6.61	Orange	2909	4.76	Black	937	1.37
White	7666	12.55	Black	2375	3.89	None	1059	1.73
Yellow	8543	13.99	None	3530	5.78	Veil color		
Blue	828	1.36	Stem Root			Brown	3732	6.11
Orange	3656	5.99	Bulbous	23906	39.15	Purple	3522	5.77
Black	1279	2.00	Swollen	19779	32.39	Red	1633	2.67
Bruise-bleed			Club	6607	10.93	White	46307	75.83
True	10590	17.34	Rooted	7529	12.33	Yellow	3740	6.12
False	50479	82.66	Rhizomorphs	3178	5.20	Black	2136	3.50
Has ring			Habitat			Class		
True	15179	24.86	Grasses	7943	13.01	Poisonous	33888	55.49
False	45890	75.14	Leaves	3168	5.19	Edible	27181	44.51

Similarly, table 3 shows a numeric summary of the remaining three quantitative variables. Results were tabulated on the training, validation, and test sets combined; no imputation was performed for these variables since they were complete in the original data.

Table 3: Numerical Summary of Quantitative Variables across all Data Sets						
Variable	**Min**	**1st Quantile**	**Median**	**Mean**	**3rd Quantile**	**Max**
Cap diameter	0.380	3.480	5.860	6.734	8.540	62.340
Stem height	0	4.640	5.950	6.582	7.740	33.920
Stem width	0	5.21	10.19	12.15	16.57	103.91

Penalized Logistic Regression

Following data cleaning, the penalized logistic regression was fit on the training set using the variables listed in tables 2 and 3. The optimal tuning parameters for the logistic regression were α = 0, corresponding to the ridge penalty, and λ = 0.01, which was the smallest penalty tuning option. To tune the cutpoint, the model was fit on the validation data. The resulting confusion matrix and model metrics are shown in tables 4 and 5. Note that the no-information rate is 0.5549, so any model with a higher accuracy would indicate that the model performed better than predicting the mode of the outcome variable.

Table 4: Confusion Matrix for Logistic Regression Performance on Validation data		
	True Edible	True Poisonous
Predicted Edible	3576	595
Predicted Poisonous	773	4827

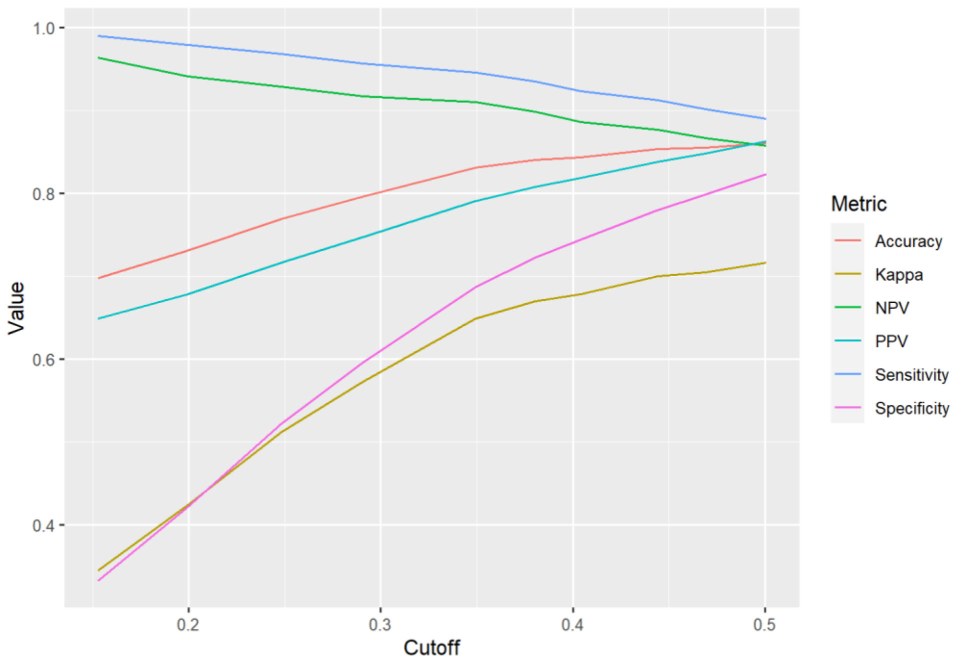
Table 5: Model Metrics for Penalized Logistic Regression Performance on Validation data	
Accuracy	0.8600
Kappa	0.7154
Sensitivity	0.8903
Specificity	0.8223
Positive Predicted Value	0.8620
Negative Predicted Value	0.8673

As mentioned in the Methods section, sensitivity is a crucial metric in the context of this data. In order to minimize the number of false negatives, the cutpoint should be lowered. Since the sensitivity on the validation data was approximately 0.89, ten equally spaced values between 0.89 and 0.99 were pre-specified as the desired sensitivities. The cutpoint required to achieve each was calculated. The potential minimum sensitivities and the corresponding cutpoints and model metrics are summarized in table 6.

Table 6: Model Performance Metrics across Potential Sensitivities on the Validation Set						
Sensitivity	**Cutpoint**	**Specificity**	**Accuracy**	**Kappa**	**PPV**	**NPV**
0.890	0.501	0.823	0.860	0.716	0.863	0.857
0.901	0.469	0.798	0.855	0.705	0.848	0.866
0.912	0.444	0.780	0.853	0.700	0.838	0.877
0.923	0.404	0.744	0.844	0.678	0.818	0.886
0.934	0.380	0.722	0.840	0.670	0.808	0.898
0.946	0.349	0.687	0.821	0.648	0.790	0.910
0.957	0.291	0.596	0.796	0.573	0.747	0.917
0.968	0.248	0.521	0.769	0.511	0.716	0.919
0.979	0.200	0.421	0.731	0.424	0.678	0.941
0.990	0.153	0.333	0.697	0.345	0.649	0.963

Figure 1 illustrates the trade-off between the increase in sensitivity and the changes in the model performance. When the cutpoint decreases from 0.349 to 0.291, there appears to be a distinct drop off in the accuracy, Kappa statistic, and specificity of the model. In order to maximize sensitivity without sacrificing the other aspects of model performance, the optimal cutpoint selected was 0.349.

Figure 1: Model Performance Metrics across Cutpoints



The final model was fit on the test data using the hyperparameters chosen during training. The confusion matrix is shown in table 7 below. The final sensitivity was 0.932, an improvement from the validation sensitivity. A more detailed summary of model metrics is shown in table 11.

Table 7: Confusion Matrix for Penalized Logistic Regression Performance on Test data		
	True Edible	True Poisonous
Predicted Edible	3844	459
Predicted Poisonous	1592	6318

A calibration plot for the final logistic regression model is shown in figure 2. The probabilities are considerably well calibrated. There is some moderate overshooting in the predicted probability around the 35% bin midpoint, where the observed event percentage was 21.16%. Likewise, there is some moderate undershooting in predicted probability around the 65% bin midpoint, where the observed event percentage is 76.03%. There are a healthy number of test observations in each of these bins, so there is no reason to believe that the plot is misleading.

Figure 2: Calibration Plot for Logistic Regression Model

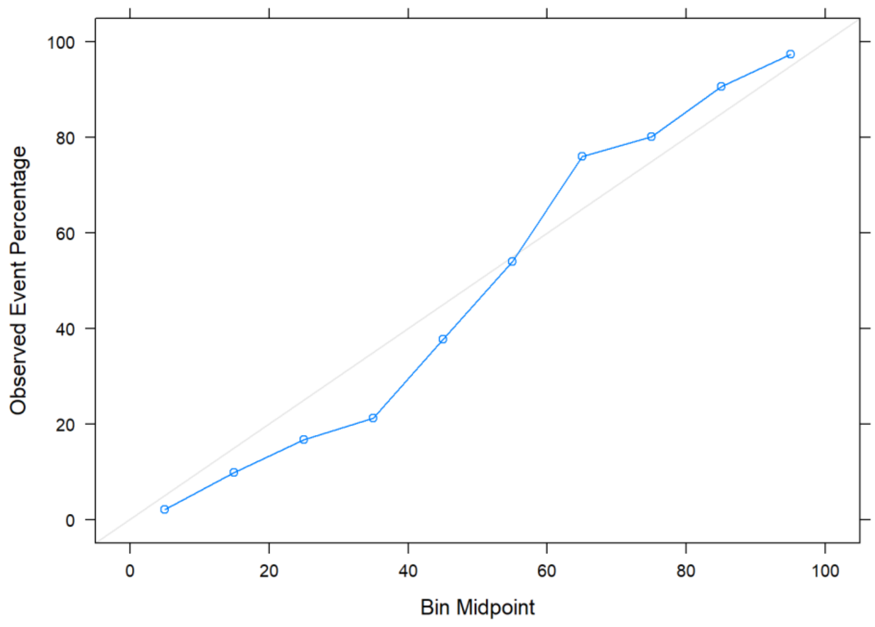


Figure 3 illustrates the receiver operating curve for the penalized logistic regression model. As demonstrated by the strong concave downward curvature, the model fit is very good. The area under the curve is 0.9284, with a 95% DeLong's confidence interval of (0.9239, 0.9328).

Figure 3: ROC Curve for Logistic Regression Model

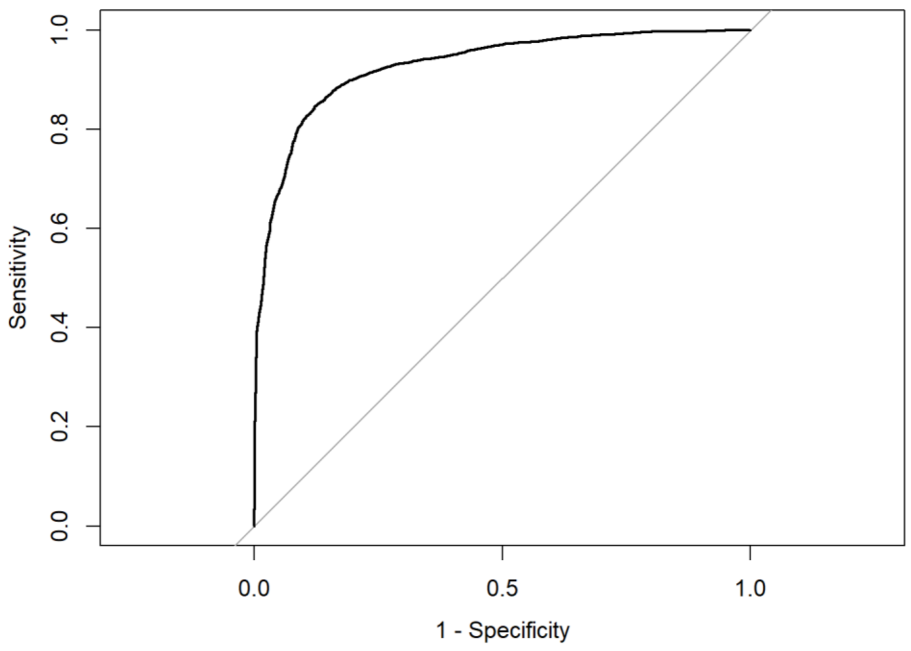
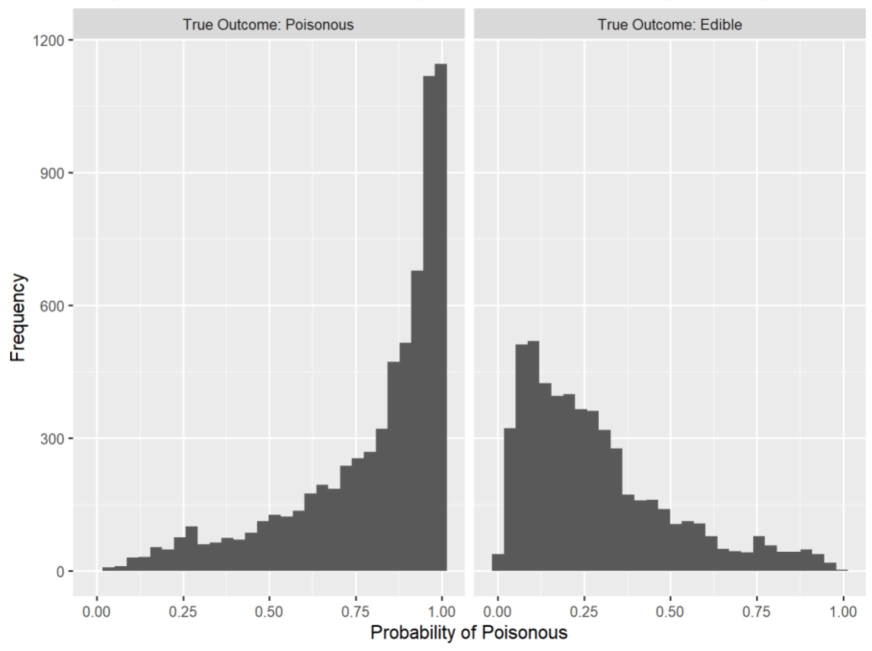


Figure 4 shows the fitted probabilities that an observation is poisonous faceted by the true class. As illustrated by the approximate bell curve across the two facets, the model did a good job of predicting each class. However, the model did a mildly better job at detecting true poisonous mushrooms than true edible mushrooms, since there is a higher frequency of poisonous mushrooms receiving probabilities near 1 than there are edible mushrooms receiving probabilities near 0.

Figure 4: Fitted Probabilities By True Outcome for Logistic Regression



Furthermore, the variable coefficients can introduce meaning to the model. The characteristics with strongly negative coefficients indicate an association between the particular characteristic and a smaller odds of being poisonous, and characteristics with larger, positive coefficients indicate an association between the particular characteristic and a larger odds of being poisonous. These characteristics are discussed in the Discussion section.

Random Forest

In addition to the penalized logistic regression model, a random forest model was fit on the training set using the variables listed in tables 2 and 3. Optimally, 3 randomly sampled predictors are considered at each split, the Gini Index is the measure of node impurity, and the minimum terminal node size is 5. The model performed extremely well on the validation data, as shown in tables 8 and 9, so the cutpoint was not tuned. While this is surprising, there were indeed no variables that were a direct function of the class outcome as described by the data dictionary.

Table 8: Confusion Matrix for Random Forest Performance on Validation data

	True Edible	True Poisonous
Predicted Edible	4329	40
Predicted Poisonous	20	5382

Table 9: Model Metrics for Random Forest Performance on Validation data

Accuracy	0.9939
Kappa	0.9876
Sensitivity	0.9926
Specificity	0.9954
Positive Predicted Value	0.9963
Negative Predicted Value	0.9908

The final model was fit on the test data using the parameters chosen during training. The confusion matrix is shown in table 10 below. As desired, there are very few false negatives.

Table 10: Confusion Matrix for Random Forest Performance on Test data

	True Edible	True Poisonous
Predicted Edible	5417	17
Predicted Poisonous	19	6760

Figure 5 shows a calibration plot for the Naive-Bayes recalibrated probabilities of the random forest model. The probabilities are well calibrated considering the minor deviations from the ideal line.

However, there are less than ten positive observations per bin for all of the bins except for the largest probability bin. Therefore, the calibration results are slightly misleading.

Figure 5: Calibration Plot for Random Forest

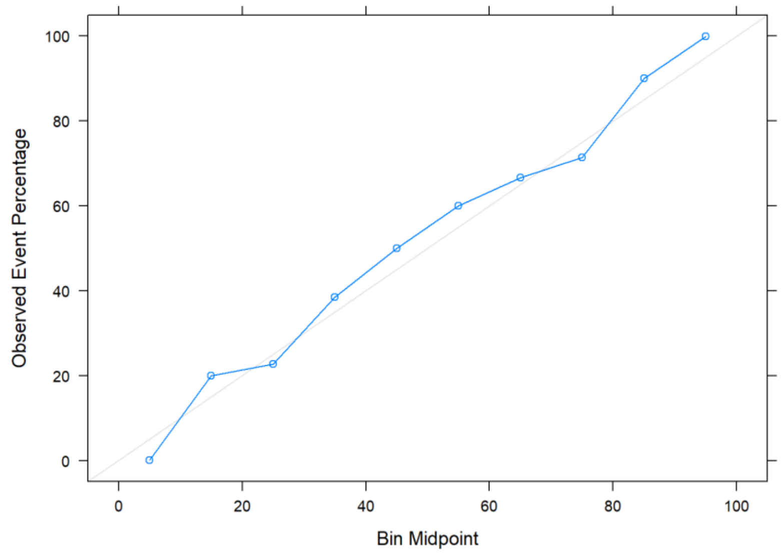


Figure 6 illustrates the receiver operating curve for the random forest model. The fit is excellent, as demonstrated by the 90-degree angle of the curve. The area under the curve is 1, with a 95% DeLong's confidence interval of (0.9999, 1).

Figure 6: ROC Curve for Random Forest Model

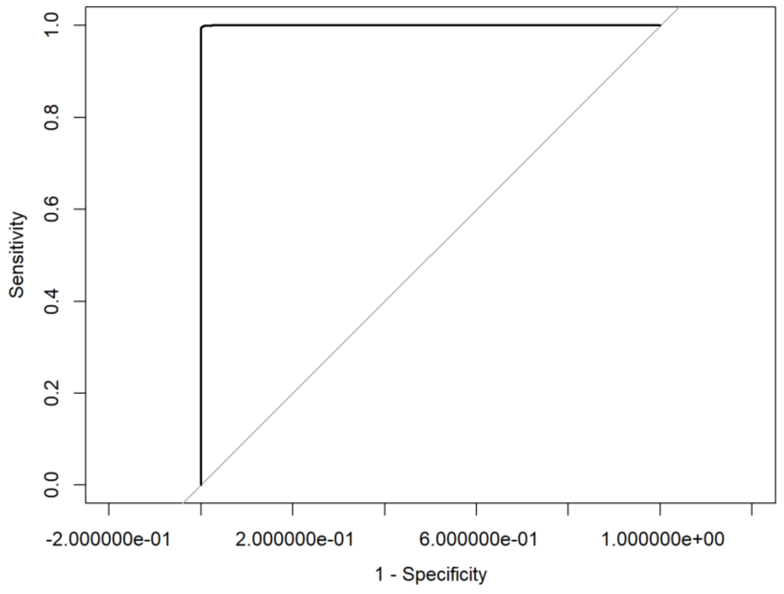


Figure 7 shows the predicted probabilities from the random forest model faceted by the true class. As indicated by the narrow left skew in the 'Poisonous' facet and the narrow right skew in the 'Edible' facet, the model did an excellent job of predicting the true class. Nearly all of the truly poisonous observations received a probability greater than 0.5, and nearly all of the truly edible observations received a probability less than 0.5.

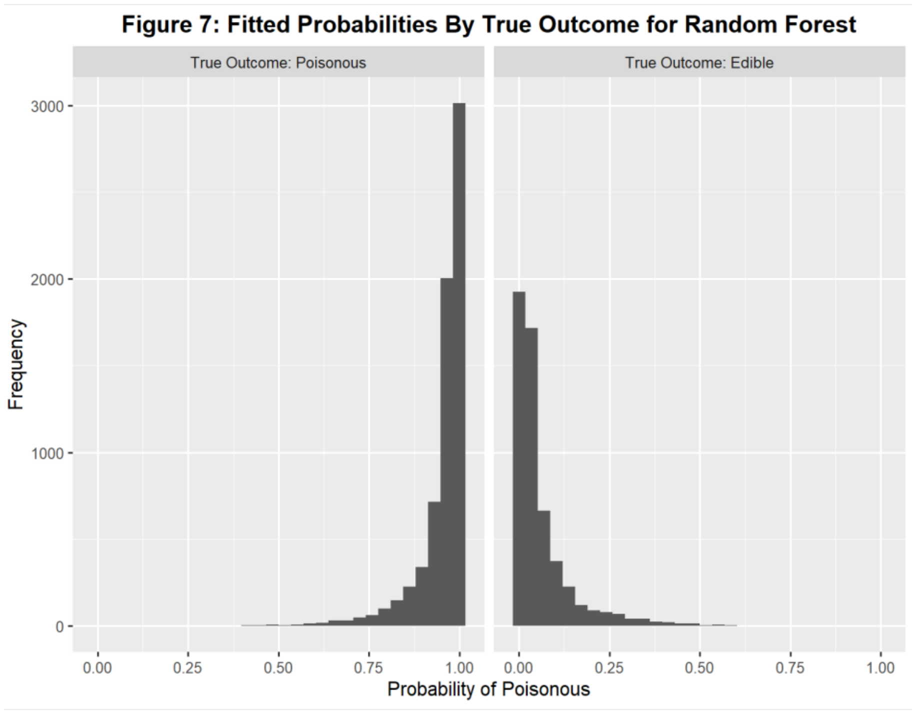
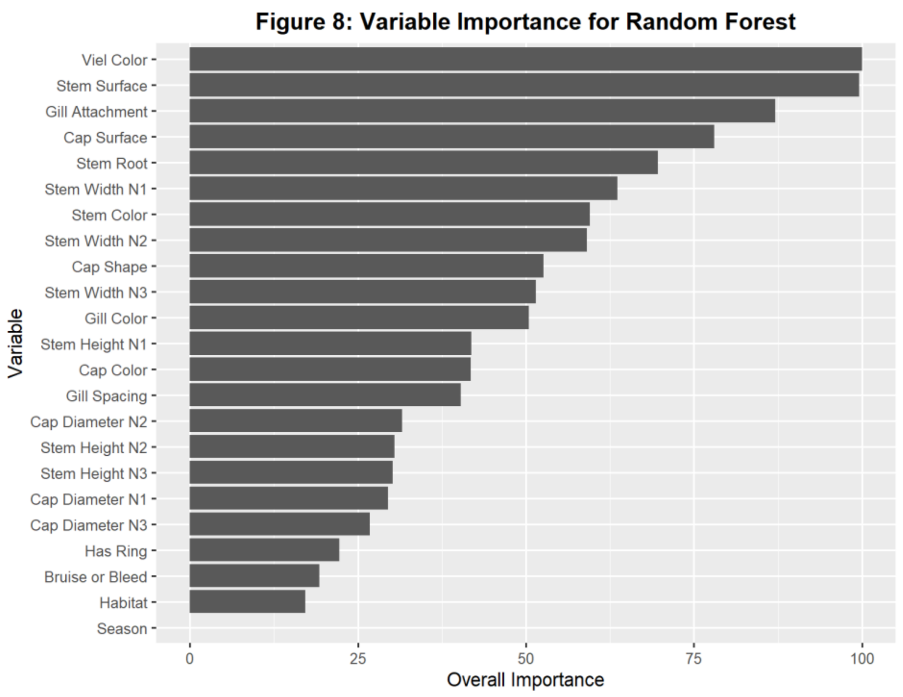


Figure 8 shows the variable importance of the random forest model with respect to predictive performance. The veil color and texture of stem surface appear to be the most important variables, followed by the gill attachment type, texture of the cap surface, stem root type, stem width, stem color, cap shape, and gill color.



Finally, table 11 shows a comparison of the model performance metrics between the logistic regression and random forest models. The random forest has extremely high values across all metrics, outperforming the penalized logistic regression model considerably.

Table 11: Model Metrics for Random Forest and Penalized Logistic Regression on Test data

Metric	Random Forest	Logistic Regression
Accuracy	0.9971	0.8321
Kappa	0.9940	0.6529
Sensitivity	0.9975	0.9323
Specificity	0.9965	0.7071
Positive Predicted Value	0.9972	0.7987
Negative Predicted Value	0.9969	0.8197
AUC	1.0000	0.9284
Brier Score	1.0076	1.0786

Discussion

Overall, the random forest model strongly outpaced the logistic regression model in terms of predictive accuracy. However, this is not meant to discredit the usefulness of the logistic regression framework; rather, the random forest was a better fit for this data in particular. Within the context of biology, there may be several different combinations of qualitative features which indicate poisonousness, and the random forest model has the advantage of being able to capture these highly complex and nonlinear relationships with the class outcome. Furthermore, it is likely that the logistic regression fit was muddied by the increased dimensionality due to the dummy binary indicator columns. A random forest does not suffer from this issue since decisions are made based on class membership.

According to the logistic regression model, mushrooms that are found on paths, have silky or green caps, or rooted, shiny, or black stems are generally more associated with poisonous specimen, on average and holding other variables constant. On the other hand, mushrooms that are found in the woods, have yellow veils, grow in the summer, have porous gills, or a conical cap shape are generally more associated with less poisonous or edible specimen, on average and holding other variables constant. According to the random forest model, veil color, stem surface texture, and gill attachment type are strong indicators of the edibility of the mushrooms; however, the particular values of these characteristics which indicate edibility are unknown.

The models have some agreement regarding veil color and gill type as strong indicators of poisonousness. However, the random forest model indicated that habitat is the second most useless variable in predicting the class outcome, whereas for the logistic regression model, mushrooms found on paths or in the woods were associated with larger and smaller odds of poisonousness, respectively.

The most notable limitation of this analysis is that the mushroom data are synthetically generated. As a result, the models perform unrealistically well. This limitation is particularly evident in the random forest model, which had nearly perfect performance on the test data. There were 193 mushroom species used to generate the data; however, there are thousands of known mushroom species. Thus, the model may only be useful for specimen from a particular geographic region from which the data were collected. Additionally, since the data are based relatively few species, it is possible that there is limited heterogeneity in the observations, resulting in clearer signal towards the class outcome.

During the cutpoint tuning process for the logistic regression model, a subjective decision was made regarding the optimal cutpoint. Firstly, the range of potential minimum sensitivities was dependent on the performance of the data on the validation set. Had the validation sensitivity been lower, a sparser but wider range of potential cutpoints would have been analyzed, perhaps leading to a different optimal result. Additionally, the trade-off between sensitivity and specificity was assessed subjectively. A more conservative or liberal analyst may have selected a higher or lower cutpoint.

In summary, the data were cleaned via k-nearest neighbor imputation, natural cubic splines were constructed for the three quantitative variables, and a penalized logistic regression model and a random forest model were built and evaluated. For this dataset, the random forest model had considerably better predictive accuracy. Perhaps random forest models are more effective than logistic regression models at classifying outcomes on datasets with many categorical variables. Ultimately, models such as these may be extremely useful for students and researchers studying mycology. Scientists may consider the characteristics discussed above when classifying various mushrooms. These frameworks may also be extended to plant biology and other ecological studies.

Appendix

```
## Final Project
## Delaney Helgeson
#####################

#Load libraries
library(VIM)
library(caret)
library(tidyr)
library(splines)
library(caret)
library(glmnet)
library(ranger)
library(OptimalCutpoints)
library(kernlab)
library(pROC)
library(dplyr)
library(klaR)

###########################
## Initial PreProcessing ##
###########################

setwd("C:\\Users\\delan\\Downloads\\STAT6302\\Final Project")

# Upload raw data
mushroom <- read.csv("secondary_data.csv", sep=";",  na.strings = "", stringsAsFactors = T)
str(mushroom)

# See which columns have missing values
colSums(is.na(mushroom))*100/dim(mushroom)[1]

# Remove spore.print.color (89.595%) and veil.type (94.798%)
mushroom_r <- subset(mushroom, select=-c(spore.print.color,veil.type))

# Remove Near Zero Variance
nearZero <- nearZeroVar(mushroom_r, saveMetrics = T)
mushroom_r[,17]
# Remove ring.type
mushroom_r2 <- mushroom_r[,-nearZero]

# Train Validation Test Rows
set.seed(425)
train_valid_Rows <- createDataPartition(mushroom_r2$class, p = 0.8, list = F)
mushroom_train_valid <- mushroom_r2[train_valid_Rows,]
set.seed(425)
train_Rows <- createDataPartition(mushroom_train_valid$class, p = 0.8, list = F)
```

```
# Create Train, Validation, and Test splits
mushroom_train <- mushroom_train_valid[train_Rows,]
mushroom_valid <- mushroom_train_valid[-train_Rows,]
mushroom_test <- mushroom_r2[-train_valid_Rows,]

# Impute Validation Set
set.seed(425)
mushroom_valid_imp <- kNN(mushroom_valid, variable=c("cap.surface", "gill.attachment",
"gill.spacing", "stem.root",
                                    "stem.surface","veil.color"), imp_var=FALSE)
# Write file to csv for later use
write.csv(mushroom_valid_imp, "mushroom_valid_imp.csv")
# Cross your fingers
colSums(is.na(mushroom_valid_imp))*100/dim(mushroom_valid_imp)[1]

# Impute Test Set
set.seed(425)
mushroom_test_imp <- kNN(mushroom_test, variable=c("cap.surface", "gill.attachment",
"gill.spacing", "stem.root",
                                   "stem.surface","veil.color"), imp_var=FALSE)

# Write file to csv for later use
write.csv(mushroom_test_imp, "mushroom_test_imp.csv")
# Cross your fingers
colSums(is.na(mushroom_test_imp))*100/dim(mushroom_test_imp)[1]

# Impute Train Set
set.seed(425)
mushroom_train_imp <- kNN(mushroom_train, variable=c("cap.surface", "gill.attachment",
"gill.spacing", "stem.root",
                                    "stem.surface","veil.color"), imp_var=FALSE)
# Write file to csv for later use
write.csv(mushroom_train_imp, "mushroom_train_imp.csv")
# Cross your fingers
colSums(is.na(mushroom_train_imp))*100/dim(mushroom_train_imp)[1]


# Used this block of code for creating the table in the report; enter the names of the
# variables to see their distribtion
mushroom_summary <- rbind(mushroom_train_imp, mushroom_valid_imp, mushroom_test_imp)
variable <- mushroom_summary %>% group_by(gill.spacing) %>% summarize(count_each = n())
variable <- data.frame(variable)
variable$percentage <- round(variable$count_each*100/ sum(variable$count_each),2)
variable
# For numeric variables
summary(mushroom_summary$cap.diameter)
```

```
#############
## Splines ##
#############

# Upload imputed data
mushroom_train_imp <- read.csv("mushroom_train_imp.csv", stringsAsFactors = T)
mushroom_valid_imp <- read.csv("mushroom_valid_imp.csv", stringsAsFactors = T)
mushroom_test_imp <- read.csv("mushroom_test_imp.csv", stringsAsFactors = T)

# Drop ID column
mushroom_train_imp <- mushroom_train_imp[,-1]
mushroom_valid_imp <- mushroom_valid_imp[,-1]
mushroom_test_imp <- mushroom_test_imp[,-1]

# Create Splines for Numeric variables (cap.diameter, stem.height, stem.width)
cap.diameter_spline <- ns(mushroom_train_imp$cap.diameter, df=3)
stem.height_spline <- ns(mushroom_train_imp$stem.height, df=3)
stem.width_spline <- ns(mushroom_train_imp$stem.width, df=3)
cap.diameter_spline_valid <- ns(mushroom_valid_imp$cap.diameter, df=3)
stem.height_spline_valid <- ns(mushroom_valid_imp$stem.height, df=3)
stem.width_spline_valid <- ns(mushroom_valid_imp$stem.width, df=3)
cap.diameter_spline_test <- ns(mushroom_test_imp$cap.diameter, df=3)
stem.height_spline_test <- ns(mushroom_test_imp$stem.height, df=3)
stem.width_spline_test <- ns(mushroom_test_imp$stem.width, df=3)

# Rename columns
colnames(cap.diameter_spline)[1:3] <- c('cap.diameter.N1', 'cap.diameter.N2', 'cap.diameter.N3')
colnames(stem.height_spline)[1:3] <- c('stem.height.N1', 'stem.height.N2', 'stem.height.N3')
colnames(stem.width_spline)[1:3] <- c('stem.width.N1', 'stem.width.N2', 'stem.width.N3')
colnames(cap.diameter_spline_valid)[1:3] <- c('cap.diameter.N1', 'cap.diameter.N2',
'cap.diameter.N3')
colnames(stem.height_spline_valid)[1:3] <- c('stem.height.N1', 'stem.height.N2', 'stem.height.N3')
colnames(stem.width_spline_valid)[1:3] <- c('stem.width.N1', 'stem.width.N2', 'stem.width.N3')
colnames(cap.diameter_spline_test)[1:3] <- c('cap.diameter.N1', 'cap.diameter.N2',
'cap.diameter.N3')
colnames(stem.height_spline_test)[1:3] <- c('stem.height.N1', 'stem.height.N2', 'stem.height.N3')
colnames(stem.width_spline_test)[1:3] <- c('stem.width.N1', 'stem.width.N2', 'stem.width.N3')

# Join to data set
mushroom_train_ns1 <- cbind(mushroom_train_imp, cap.diameter_spline, stem.height_spline,
stem.width_spline)
mushroom_valid_ns1 <- cbind(mushroom_valid_imp, cap.diameter_spline_valid,
stem.height_spline_valid, stem.width_spline_valid)
mushroom_test_ns1 <- cbind(mushroom_test_imp, cap.diameter_spline_test,
stem.height_spline_test, stem.width_spline_test)
# Drop original variables
mushroom_train_ns <- subset(mushroom_train_ns1, select = -c(cap.diameter, stem.height,
stem.width))
```

```
mushroom_valid_ns <- subset(mushroom_valid_ns1, select = -c(cap.diameter, stem.height,
stem.width))
mushroom_test_ns <- subset(mushroom_test_ns1, select = -c(cap.diameter, stem.height,
stem.width))

######################################
## Logistic Regression: Tune Model ##
######################################

# Define predictors/response split
X_train <- model.matrix(class ~., mushroom_train_ns) # dummy encoded
y_train <- mushroom_train_ns[,1]
X_valid <- model.matrix(class ~., mushroom_valid_ns)
y_valid <- mushroom_valid_ns[,1]
y_valid <- ifelse(y_valid == 'e', 0, 1)
X_test <- model.matrix(class ~., mushroom_test_ns)
y_test <- mushroom_test_ns[,1]
y_test <- ifelse(y_test == 'e', 0, 1)

# Remove R objects to save space
rm(mushroom_train_imp)
rm(mushroom_valid_imp)
rm(mushroom_test_imp)
rm(mushroom_train_ns1)
rm(mushroom_valid_ns1)
rm(mushroom_test_ns1)
rm(cap.diameter_spline)
rm(stem.height_spline)
rm(stem.width_spline)
rm(cap.diameter_spline_valid)
rm(stem.height_spline_valid)
rm(stem.width_spline_valid)
rm(cap.diameter_spline_test)
rm(stem.height_spline_test)
rm(stem.width_spline_test)


# hyperparameter grid
glmnGrid <- expand.grid(alpha = c(0, 0.5, 1),
                    lambda = seq(0.01, 0.2, length = 10))

# Set train control: 10-fold cross validation repeated five times.
ctrl <- trainControl(method = "repeatedcv",
                classProbs = TRUE,
                repeats=5,
                savePredictions = TRUE)
# Train model
set.seed(425)
```

```
Logistic_glm <- train(x = X_train,
                  y = y_train,
                  method = "glmnet",
                  tuneGrid = glmnGrid,
                  metric = "Accuracy",
                  family = "binomial",
                  trControl = ctrl)
saveRDS(Logistic_glm, "Logistic_glm.rds")

# Optimal Parameters and results
Logistic_glm$bestTune
Logistic_glm$results

#####################
## Cutpoint Tuning ##
#####################

# Read in model
Logistic_glm <- readRDS("Logistic_glm.rds")

# Coefficients of final model
coef(Logistic_glm$finalModel, s=Logistic_glm$bestTune$lambda)

# Preds: Logistic regression on Validation data; Used for choosing a cutpoint
logistic_predict_valid <- predict(Logistic_glm$finalModel, newx=X_valid,
s=Logistic_glm$bestTune$lambda, type='response')

# Logistic Validation Results
Logistic_results <- cbind(y_valid, logistic_predict_valid)
Logistic_results <- data.frame(Logistic_results)
Logistic_results$y_valid <- factor(Logistic_results$y_valid)
Logistic_results$predclass <- factor(ifelse(Logistic_results$s1 > 0.5, 1, 0))

# See what is the sensitivity of un-adjusted cut-point model
confusionMatrix(data = Logistic_results$predclass,
             reference = Logistic_results$y_valid, positive="1")

# Tune model for optimal cut point
SeValues <- seq(0.89, 0.99, length = 10)

# Test all desired sensitivities: Find a balance between Accuracy, Sensitivity and Specificity
# It's not letting me access the optimal.cutpoints objects from a list properly... so we will do it
individually
# Se = 0.89
Opt.cpt.1 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                    control = control.cutpoints(valueSe = SeValues[1]))
# Se = 0.90
```

```
Opt.cpt.2 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                    control = control.cutpoints(valueSe = SeValues[2]))
# Se = 0.91
Opt.cpt.3 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                    control = control.cutpoints(valueSe = SeValues[3]))
# Se = 0.92
Opt.cpt.4 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                    control = control.cutpoints(valueSe = SeValues[4]))
# Se = 0.93
Opt.cpt.5 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                    control = control.cutpoints(valueSe = SeValues[5]))
# Se = 0.945
Opt.cpt.6 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                    control = control.cutpoints(valueSe = SeValues[6]))
# Se = 0.956
Opt.cpt.7 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                    control = control.cutpoints(valueSe = SeValues[7]))
# Se = 0.967
Opt.cpt.8 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                    control = control.cutpoints(valueSe = SeValues[8]))
# Se = 0.978
Opt.cpt.9 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                    control = control.cutpoints(valueSe = SeValues[9]))
# Se = 0.99
Opt.cpt.10 <- optimal.cutpoints(X="s1", status="y_valid", data=data.frame(Logistic_results),
tag.healthy='0', methods='MinValueSe',
                     control = control.cutpoints(valueSe = SeValues[10]))

summary(Opt.cpt.1)
summary(Opt.cpt.2)
summary(Opt.cpt.3)
summary(Opt.cpt.4)
summary(Opt.cpt.5)
summary(Opt.cpt.6)
summary(Opt.cpt.7)
summary(Opt.cpt.8)
summary(Opt.cpt.9)
summary(Opt.cpt.10)

Specificities <- c(summary(Opt.cpt.1)$p.table$Global$MinValueSe[[1]]["Sp",],
              summary(Opt.cpt.2)$p.table$Global$MinValueSe[[1]]["Sp",],
              summary(Opt.cpt.3)$p.table$Global$MinValueSe[[1]]["Sp",],
              summary(Opt.cpt.4)$p.table$Global$MinValueSe[[1]]["Sp",],
              summary(Opt.cpt.5)$p.table$Global$MinValueSe[[1]]["Sp",],
              summary(Opt.cpt.6)$p.table$Global$MinValueSe[[1]]["Sp",],
              summary(Opt.cpt.7)$p.table$Global$MinValueSe[[1]]["Sp",],
              summary(Opt.cpt.8)$p.table$Global$MinValueSe[[1]]["Sp",],
              summary(Opt.cpt.9)$p.table$Global$MinValueSe[[1]]["Sp",],
              summary(Opt.cpt.10)$p.table$Global$MinValueSe[[1]]["Sp",])

Cutoff <- c(summary(Opt.cpt.1)$p.table$Global$MinValueSe[[1]]["cutoff",],
          summary(Opt.cpt.2)$p.table$Global$MinValueSe[[1]]["cutoff",],
          summary(Opt.cpt.3)$p.table$Global$MinValueSe[[1]]["cutoff",],
          summary(Opt.cpt.4)$p.table$Global$MinValueSe[[1]]["cutoff",],
          summary(Opt.cpt.5)$p.table$Global$MinValueSe[[1]]["cutoff",],
          summary(Opt.cpt.6)$p.table$Global$MinValueSe[[1]]["cutoff",],
          summary(Opt.cpt.7)$p.table$Global$MinValueSe[[1]]["cutoff",],
          summary(Opt.cpt.8)$p.table$Global$MinValueSe[[1]]["cutoff",],
          summary(Opt.cpt.9)$p.table$Global$MinValueSe[[1]]["cutoff",],
          summary(Opt.cpt.10)$p.table$Global$MinValueSe[[1]]["cutoff",])

SeTuningMetrics <- data.frame(cbind(SeValues, Specificities, Cutoff))
colnames(SeTuningMetrics)[1:2] <- c("Sensitivity", "Specificity")

# Append outcomes to Logistic results of different cutpoints
Logistic_results$predclass_se1 <- factor(ifelse(Logistic_results$s1 > Cutoff[1], 1, 0))
Logistic_results$predclass_se2 <- factor(ifelse(Logistic_results$s1 > Cutoff[2], 1, 0))
Logistic_results$predclass_se3 <- factor(ifelse(Logistic_results$s1 > Cutoff[3], 1, 0))
Logistic_results$predclass_se4 <- factor(ifelse(Logistic_results$s1 > Cutoff[4], 1, 0))
Logistic_results$predclass_se5 <- factor(ifelse(Logistic_results$s1 > Cutoff[5], 1, 0))
Logistic_results$predclass_se6 <- factor(ifelse(Logistic_results$s1 > Cutoff[6], 1, 0))
Logistic_results$predclass_se7 <- factor(ifelse(Logistic_results$s1 > Cutoff[7], 1, 0))
Logistic_results$predclass_se8 <- factor(ifelse(Logistic_results$s1 > Cutoff[8], 1, 0))
Logistic_results$predclass_se9 <- factor(ifelse(Logistic_results$s1 > Cutoff[9], 1, 0))
Logistic_results$predclass_se10 <- factor(ifelse(Logistic_results$s1 > Cutoff[10], 1, 0))

# Construct Confusion Matrices to Extract Metrics
conf_matrix1 <- confusionMatrix(data = Logistic_results$predclass_se1,
                         reference = Logistic_results$y_valid, positive="1")
conf_matrix2 <- confusionMatrix(data = Logistic_results$predclass_se2,
                         reference = Logistic_results$y_valid, positive="1")
conf_matrix3 <- confusionMatrix(data = Logistic_results$predclass_se3,
                         reference = Logistic_results$y_valid, positive="1")
conf_matrix4 <- confusionMatrix(data = Logistic_results$predclass_se4,
                         reference = Logistic_results$y_valid, positive="1")
conf_matrix5 <- confusionMatrix(data = Logistic_results$predclass_se5,
                         reference = Logistic_results$y_valid, positive="1")
conf_matrix6 <- confusionMatrix(data = Logistic_results$predclass_se6,
                         reference = Logistic_results$y_valid, positive="1")
conf_matrix7 <- confusionMatrix(data = Logistic_results$predclass_se7,
                         reference = Logistic_results$y_valid, positive="1")
conf_matrix8 <- confusionMatrix(data = Logistic_results$predclass_se8,
                         reference = Logistic_results$y_valid, positive="1")
conf_matrix9 <- confusionMatrix(data = Logistic_results$predclass_se9,
                         reference = Logistic_results$y_valid, positive="1")
conf_matrix10 <- confusionMatrix(data = Logistic_results$predclass_se10,
                          reference = Logistic_results$y_valid, positive="1")

SeTuningMetrics$Kappa <- c(conf_matrix1$overall["Kappa"], conf_matrix2$overall["Kappa"],
conf_matrix3$overall["Kappa"],
                     conf_matrix4$overall["Kappa"], conf_matrix5$overall["Kappa"],
conf_matrix6$overall["Kappa"],
                     conf_matrix7$overall["Kappa"], conf_matrix8$overall["Kappa"],
conf_matrix9$overall["Kappa"],
                     conf_matrix10$overall["Kappa"])

SeTuningMetrics$Accuracy <- c(conf_matrix1$overall["Accuracy"], conf_matrix2$overall["Accuracy"],
conf_matrix3$overall["Accuracy"],
                        conf_matrix4$overall["Accuracy"], conf_matrix5$overall["Accuracy"],
conf_matrix6$overall["Accuracy"],
                        conf_matrix7$overall["Accuracy"], conf_matrix8$overall["Accuracy"],
conf_matrix9$overall["Accuracy"],
                        conf_matrix10$overall["Accuracy"])

SeTuningMetrics$PPV <- c(conf_matrix1$byClass["Pos Pred Value"], conf_matrix2$byClass["Pos Pred
Value"],
                   conf_matrix3$byClass["Pos Pred Value"], conf_matrix4$byClass["Pos Pred Value"],
                   conf_matrix5$byClass["Pos Pred Value"], conf_matrix6$byClass["Pos Pred Value"],
                   conf_matrix7$byClass["Pos Pred Value"], conf_matrix8$byClass["Pos Pred Value"],
                   conf_matrix9$byClass["Pos Pred Value"], conf_matrix10$byClass["Pos Pred Value"])

SeTuningMetrics$NPV <- c(conf_matrix1$byClass["Neg Pred Value"], conf_matrix2$byClass["Neg Pred
Value"],
                   conf_matrix3$byClass["Neg Pred Value"], conf_matrix4$byClass["Neg Pred Value"],
                   conf_matrix5$byClass["Neg Pred Value"], conf_matrix6$byClass["Neg Pred Value"],
                   conf_matrix7$byClass["Neg Pred Value"], conf_matrix8$byClass["Neg Pred Value"],
                   conf_matrix9$byClass["Neg Pred Value"], conf_matrix10$byClass["Neg Pred Value"])

# Metrics dataframe for various cutoffs
SeTuningMetrics
#write.csv(SeTuningMetrics, "SeTuningMetrics.csv")
SeTuningMetrics_long <- gather(SeTuningMetrics, Metric, Value, c(1,2,4,5,6,7))

# Plot Model Metrics for Various cutoffs
SeTuningPlot <- ggplot(SeTuningMetrics_long, aes(x = Cutoff, y = Value, colour = Metric)) +
geom_line() +
  ggtitle("Figure 1: Model Performance Metrics across Cutpoints")+
  theme(plot.title=element_text(hjust=0.5, face='bold'))
SeTuningPlot

# Grab optimal cutoff
SeTuningMetrics <- read.csv("SeTuningMetrics.csv")
opt_cpt_final <- SeTuningMetrics$Cutoff[6]

############################################
## Model Evaluation: Logistic Regression ##
############################################

# Read model to save time
Logistic_glm_final <- readRDS("Logistic_glm.rds") # same model as before, just gave it a different
name

# Predict on test data
logistic_predict_test <- predict(Logistic_glm_final$finalModel, newx=X_test,
                        s=Logistic_glm_final$bestTune$lambda, type='response')
# Logistic Validation Results
Logistic_results_final <- data.frame(cbind(y_test, logistic_predict_test))
Logistic_results_final$y_test <- factor(Logistic_results_final$y_test)
Logistic_results_final$predclass <- factor(ifelse(Logistic_results_final$s1 > opt_cpt_final, 1, 0))

## Calibration analysis
Cal_Logistic <- calibration(y_test ~ s1, data = Logistic_results_final, cuts = 10, class="1")
xyplot(Cal_Logistic, auto.key = list(columns = 2), main="Figure 2: Calibration Plot for Logistic
Regression Model")
Cal_Logistic$data

# ROC Plot
ROC_Logistic <- roc(Logistic_results_final$y_test, Logistic_results_final$s1)

# AUC
auc(ROC_Logistic)
# AUC confidence interval
ci.auc(ROC_Logistic)

# Plot ROC
plot(ROC_Logistic, legacy.axes = TRUE, main="Figure 3: ROC Curve for Logistic Regression Model")

# Confusion Matrix
confusionMatrix(data = Logistic_results_final$predclass,
          reference = Logistic_results_final$y_test, positive="1")

# Calculate brier score
(1/length(Logistic_results_final$s1)) * sum((Logistic_results_final$s1 -
as.numeric(Logistic_results_final$y_test))^2)
```

```
# Predicted Probabilities Faceted by true outcome
ggnames <- c("1" = "True Outcome: Poisonous", "0" = "True Outcome: Edible")
hist_Logistic <- ggplot(data=Logistic_results_final, aes(x=s1)) + geom_histogram() +
  facet_grid(~factor(y_test, levels = c("1", "0")),
           labeller = as_labeller(ggnames)) + xlab("Probability of Poisonous") + ylab("Frequency") +
  ggtitle("Figure 4: Fitted Probabilities By True Outcome for Logistic Regression")+
  theme(plot.title=element_text(hjust=0.5, face='bold'))
hist_Logistic

coef(Logistic_glm_final$finalModel, s=Logistic_glm_final$bestTune$lambda)

########################################
## Random Forest: Tune Initial Model ##
########################################

# Specify tuning grid
rf_grid <- expand.grid(mtry = c(3, 5, 7),
                    splitrule = c("gini", "hellinger"),
                    min.node.size = c(1, 3, 5))
# train model
set.seed(425)
rf_fit_caret <- train(x = mushroom_train_ns[,-1],
                   y = mushroom_train_ns[,1],
                   method = "ranger",
                   trControl = trainControl(method="repeatedcv", number = 10, classProbs = T),
                   metric = "Kappa",
                   tuneGrid = rf_grid,
                   num.trees = 500,
                   importance="impurity"
)
saveRDS(rf_fit_caret, "rf_fit_caret.rds")
rf_fit_caret <- readRDS("rf_fit_caret.rds")

# Results and best tuning parameters
rf_fit_caret$results
rf_fit_caret$bestTune

# Predict on validation set
rf_predict <- predict(rf_fit_caret$finalModel, data=mushroom_valid_ns[,-1],
s1=rf_fit_caret$bestTune$mtry,
                   s2=rf_fit_caret$bestTune$splitrule, s3=rf_fit_caret$bestTune$min.node.size)

# RF Validation Results
Rf_results <- data.frame(cbind(y_valid, rf_predict$predictions[,"p"]))
Rf_results$y_valid <- factor(Rf_results$y_valid)
Rf_results$predclass <- factor(ifelse(Rf_results$V2 > 0.5, 1, 0))
str(Rf_results)
```

```
# See what is the sensitivity of un-adjusted cut-point model
confusionMatrix(data = Rf_results$predclass,
          reference = Rf_results$y_valid, positive="1")
# Model performed suspiciously well, no need to tune the cutpoint.

# Investigation.. for funsies
qualities_of_poison <- cbind(mushroom_test_ns$veil.color, mushroom_test_ns$stem.surface,
data.frame(Rf_results_final$predclass))
qualities_of_poison[which(qualities_of_poison$Rf_results_final.predclass==0),1]


######################################
## Model Evaluation: Random Forest ##
######################################

# Predict on test set
rf_predict_test <- predict(rf_fit_caret$finalModel, data=mushroom_test_ns[,-1],
s1=rf_fit_caret$bestTune$mtry,
                s2=rf_fit_caret$bestTune$splitrule, s3=rf_fit_caret$bestTune$min.node.size)


# RF Test Results
Rf_results_final <- data.frame(cbind(y_test, rf_predict_test$predictions[,'p']))
Rf_results_final$y_test <- factor(Rf_results_final$y_test)
Rf_results_final$predclass <- factor(ifelse(Rf_results_final$V2 > 0.5, 1, 0))

## Calibration analysis
Cal_Rf <- calibration(y_test ~ V2, data = Rf_results_final, cuts = 10, class="1")
xyplot(Cal_Rf, auto.key = list(columns = 2), main="Figure 5: Calibration Plot for Random Forest")
Cal_Rf$data

# Recalibrate
Cal_Rf_recal <- NaiveBayes(y_test ~ V2, data = Rf_results_final, usekernel = TRUE)
lrCal <- glm(y_test ~ V2, data = Rf_results_final, family = binomial)
Rf_results_final$recalibrated <- predict(Cal_Rf_recal, Rf_results_final[,"V2",drop=F])$posterior[,2]

# RF Test Results
Rf_results_final_re <- data.frame(cbind(y_test, Rf_results_final$recalibrated))
Rf_results_final_re$y_test <- factor(Rf_results_final_re$y_test)
Cal_Rf_re_plot <- calibration(y_test ~ V2, data = Rf_results_final_re, cuts = 10, class="1")
xyplot(Cal_Rf_re_plot, auto.key = list(columns = 2), main="Figure 5: Calibration Plot for Random
Forest")
Cal_Rf_re_plot$data # When you call data, the counts column shows the number of positive counts.

# ROC
ROC_Rf <- roc(Rf_results_final$y_test, Rf_results_final$V2)
```

```
# AUC
auc(ROC_Rf)
# AUC confidence interval
ci.auc(ROC_Rf)

# Plot ROC
plot(ROC_Rf, legacy.axes = TRUE, main="Figure 6: ROC Curve for Random Forest Model")



# Confusion matrix
confusionMatrix(data = Rf_results_final$predclass,
            reference = Rf_results_final$y_test, positive="1")

# Calculate brier score
(1/length(Rf_results_final$V2)) * sum((Rf_results_final$V2 - as.numeric(Rf_results_final$y_test))^2)

# Predicted Probabilities faceted by true outcome
ggnames <- c("1" = "True Outcome: Poisonous", "0" = "True Outcome: Edible")
hist_Rf <- ggplot(data=Rf_results_final, aes(x=V2)) + geom_histogram() +
  facet_grid(~factor(y_test, levels = c("1", "0")),
          labeller = as_labeller(ggnames)) + xlab("Probability of Poisonous") + ylab("Frequency") +
  ggtitle("Figure 7: Fitted Probabilities By True Outcome for Random Forest")+
  theme(plot.title=element_text(hjust=0.5, face='bold'))
hist_Rf

# Variable Importance
varImp(rf_fit_caret)
Rf_importance <- data.frame(varImp(rf_fit_caret)$importance)

# Rename for plot
rownames(Rf_importance)[1:23] <- c("Cap Shape", "Cap Surface", "Cap Color", "Bruise or Bleed", "Gill
Attachment",
                          "Gill Spacing", "Gill Color", "Stem Root", "Stem Surface", "Stem Color", "Viel
Color",
                          "Has Ring", "Habitat", "Season", "Cap Diameter N1", "Cap Diameter N2", "Cap
Diameter N3",
                          "Stem Height N1", "Stem Height N2", "Stem Height N3", "Stem Width N1", "Stem
Width N2",
                          "Stem Width N3")
# Variable importance plot
Rf_importance %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  arrange(Overall) %>%
  mutate(rowname = forcats::fct_inorder(rowname )) %>%
  ggplot()+
  geom_col(aes(x = rowname, y = Overall))+
  ggtitle("Variable Importance for Random Forest")+
  xlab("Variable")+
  ylab("Overall Importance")+
  coord_flip()+
  theme(plot.title=element_text(hjust=0.5, face='bold'))
```