Damon Shorty
Nick Wetter
DATA 498A
5/9/23

Edible and Poisonous Mushrooms

Introduction

There are a variety of mushrooms that can be found in many places around the world. Mushrooms are essentially a fungi, which is a part of the Fungus kingdom in Biology. According to *Encyclopedia Britannica*, "Fungi are everywhere in very large numbers - in the soil and the air, in lakes, rivers, and seas, on and within plants and animals, in food and clothing, and in the human body (2023)." Fungi, like mushrooms, play an important role in continuing the nutrient cycle in different ecosystems around the world. Mushrooms can be eaten, although not all mushrooms are edible (some are poisonous). According to *The Nutrition Source*, "Mushrooms are also recognized by chefs for their ability to create savory rich flavors called umami, thanks to the presence of an amino acid called glutamate, which is also found in meats, fish, cheeses, and simmering soups." We often can find mushrooms in the wild/nature and may wonder whether a random mushroom is edible or poisonous. Could we determine whether a mushroom is edible or poisonous based on its physical characteristics? That is the question we are interested in answering for this study.

Description of Dataset

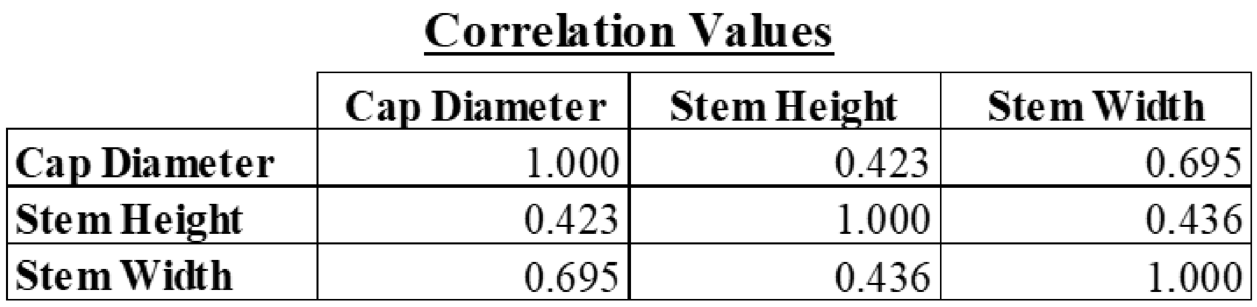
The dataset we are using in this study to determine whether a mushroom is edible or poisonous is from the UCI Machine Learning Repository. This dataset is titled the Secondary Mushroom Dataset and we have (re)named the dataset as mushrooms in our study/project for naming convenience. The mushrooms dataset has 61,069 rows and 21 columns. There is one response variable (qualitative variable) and twenty predictor variables (three quantitative and seventeen qualitative variables with levels ranging from two to thirteen). We found quite a few missing values in the mushrooms dataset that were related to a different number of qualitative (predictor) variables. To keep our data consistent, we decided to label these blank/missing values (expressed as "" in the data) to a factor level called v (expressed as "v" in the data) for any blank/missing values found throughout the qualitative variables. Also, we found an unnamed factor level "f" in the stem root variable (which is highlighted in *Figure F* within the Appendix) that we decided to keep for the sake of data consistency. The coding required for cleaning up this data can be seen in the Appendix B and the final dimensions for the mushrooms dataset are 61,069 by 21.

The response variable is class, which classifies whether or not a mushroom is edible (e=edible) or poisonous (p=poisonous). The explanatory/predictor variables include both quantitative and qualitative variables. The three quantitative variables are the following: cap.diameter (in cm), stem.height (in cm), and stem.width (in mm). The seventeen qualitative variables are the following: cap.shape (7 factor levels), cap.surface (12 factor levels), cap.color (12 factor levels), does.bruise.bleed (2 factor levels), gill.attachment (8 factor levels), gill.spacing (4 factor levels), gill.color (12 factor levels), stem.root (6 factor levels), stem.surface (9 factor levels), stem.color (13 factor levels), veil.type (2 factor levels), veil.color (7 factor levels), has.ring (2 factor levels), ring.type (9 factor levels), spore.print.color (8 factor levels), habitat (8 factor levels), and season (4 factor levels). The categorical variables are many and each has a few factor levels that are further clarified in *Figure F* in the Appendix A.

Exploring the Data

The summary/description of statistics for the mushrooms dataset includes those mentioned in this section. This section is broken down into two parts: one for the quantitative variables and one for qualitative variables. The correlation values for the quantitative variables are shown in *Figure A* below. To visualize any trends in the qualitative variables, we plotted all the qualitative variables against one another. The class (response) variable was used as a color key to show which data points on the sixteen plots produced are associated with an edible or poisonous mushroom. All of the coding for these sixteen plots can be seen in Appendix B. *Figure B* showcases one out of the sixteen plots produced to highlight the fact that there are some noticeable trends in the qualitative data. In *Figure B*, it can be seen that if a mushroom has a flat cap shape and is in an urban (or waste) habitat, then it is more than likely that the mushroom is edible. In another instance you can see that every mushroom found growing on paths was poisonous, regardless of cap shape. There are other combinations of levels (e.g., convex cap shape/heaths habitat, sunken cap shape/grasses habitat, etc.) that can be hard to determine if a mushroom is edible or poisonous by visualization alone. Further analysis in the study will help to determine how significant each predictor variable (and associated levels) are in determining whether a mushroom is edible or poisonous.

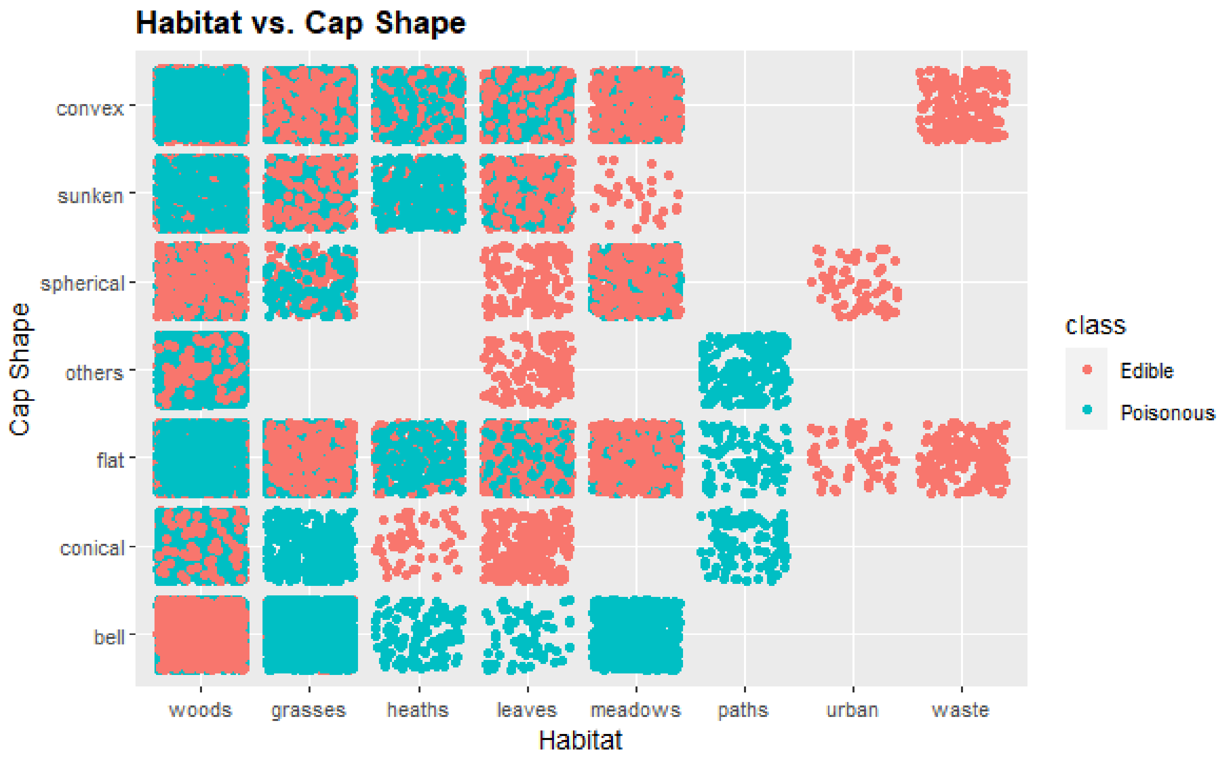
Figure A



Correlation Values

	Cap Diameter	Stem Height	Stem Width
Cap Diameter	1.000	0.423	0.695
Stem Height	0.423	1.000	0.436
Stem Width	0.695	0.436	1.000

Figure B



Review of Approaches

The approaches we used in this study included Decision Tree, Naive Bayes, K-Nearest Neighbor (KNN), Histogram-based Gradient Boosting Tree, Neural Network, and Support Vector Machine (SVM) classifiers. These approaches were chosen because the question of interest for this study has to do with a binary variable which calls for various classification approaches to determine whether a mushroom is edible or poisonous. The Decision Tree, Naive Bayes, and KNN were the most obvious approaches to use but other approaches were used as well to get a better overall understanding of the question we are trying to answer. For all of the approaches used in this study, we used a 75/25 split of the mushrooms dataset for our training data and testing data.

The Decision Tree approach for classification was used to predict the qualitative response variable for classifying edible or poisonous mushrooms. The Decision Tree can handle the qualitative predictors (mentioned in the Description of Dataset section) without the need to create dummy variables, which was helpful in our study due to the high number of qualitative predictor variables. The models for an unpruned tree and pruned tree were constructed for comparison. Methods like bagging and random forest were used to assess the predictive performance of the various decision trees. Through the use of random forest, we can also assess the performance of both the quantitative and qualitative predictor variables to determine an order for variable importance. We will use the mean decrease in Gini index to determine the order for variable importance. The Naive Bayes approach for classification was a good choice to use because we have a large p (p=20) and it has a wide range of settings. Since p is large, the other multivariate models like Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) would break down. This is the reason why we chose to use the Naive Bayes approach over the other two approaches (LDA & QDA).

When training most of the python classifiers (KNN, SVC, Neural Network) the models included in the sklearn package required that we impute missing values in the data used to train and test the models. For each model the numerical variables were imputed as mean and median of their respective columns; there was no difference between model accuracy for any of the trained models when switching between these two imputation strategies. For the factor variables, the empty values were simply replaced with the most common value in the column. For the KNN classifier we tested the values 1-100 for k, with 1 being the value with the highest associated test accuracy score. We chose to train a Histogram-based Gradient Boosting Classifier because in the sklearn package this model has support for missing values. While we were producing good results from models with the imputed values, we wanted to have at least one model that did not operate on data we had made any changes to through imputation. We tried an SVM classifier from the sklearn library as well because a perfect accuracy score was achieved in R with this model. We used the value 1 for both the gamma and C hyperparameters, which were the most accurate of all tested values. Finally, we trained a Neural Network model on our dataset. We wanted to include a model that was not a part of the coursework and it turned out to be one of our most accurate models of the entire project.

Results from Approaches

The results from the various approaches we tried in the study are outlined in this section. The results for the Decision Tree, Naive Bayes, and SVM approaches can be seen in *Figure C*. The tree plots for the unpruned tree and pruned tree can be seen in *Figure G* and *Figure H* in the Appendix A. Both tree plots appear to be similar as the resulting pruned tree had the same number of trees (28) as the unpruned tree's number of terminal nodes (28). The resulting test error rates were also the same at 0.08763. *Figure D* confirms that the number of trees with the lowest Cross-Validation (CV) Error is 28 trees. The test error rate for the Naive Bayes approach was 0.2572, which was higher than the test error rates for the Decision Tree (all methods) and SVM approaches. The bagging method (under the Decision Tree approach) had a test error rate of 0.0002 and the random forest method (under the Decision Tree approach) had a test error rate of 0. In using the random forest method, we were able to determine which predictor variables were most important in modeling our data. The top three predictor variables include Cap Shape, Gill Attachment, and Gill Color. The mean decrease Gini Index plot can be seen in *Figure E*, which shows an order for variable importance. The parameters for the SVM (with a radial kernel) method are gamma = 1 and cost = 1. The SVM (with a radial kernel) method resulted in a test error rate of 0.00013 for the corresponding parameters. The values for the parameters (gamma & cost) were verified to be the best parameters for this SVM (with a radial kernel) method by performing CV as seen in Appendix A. The test error rate after "tuning" the SVM (with a radial kernel) model was 0.00013. This test error rate was very similar to the test error rate of the SVM (with a radial kernel) model without "tuning."

The results for the KNN and Histogram-based Gradient Boosting models were actually the same; we found this interesting because one of them considered imputed values and the other never even got a value at all in those cases. Both models achieved a test accuracy value of 99.96% correct classifications. For the KNN model we tested the values 1-100 for k and produced this most accurate result with a value of k = 1. With the SVC we noticed very different results depending on the kernel used. When I trained the model using the default sklearn kernel, a test accuracy of 91.87% was achieved while using a linear kernel we got a score of 68.80%. By far the best SVC model was using the radial kernel which produced a test accuracy score of 98.16%. The Neural Network was the most accurate of all the python models, achieving a test accuracy score of 99.99% correct classifications when using the hyperparameter set mentioned previously.

Figure C

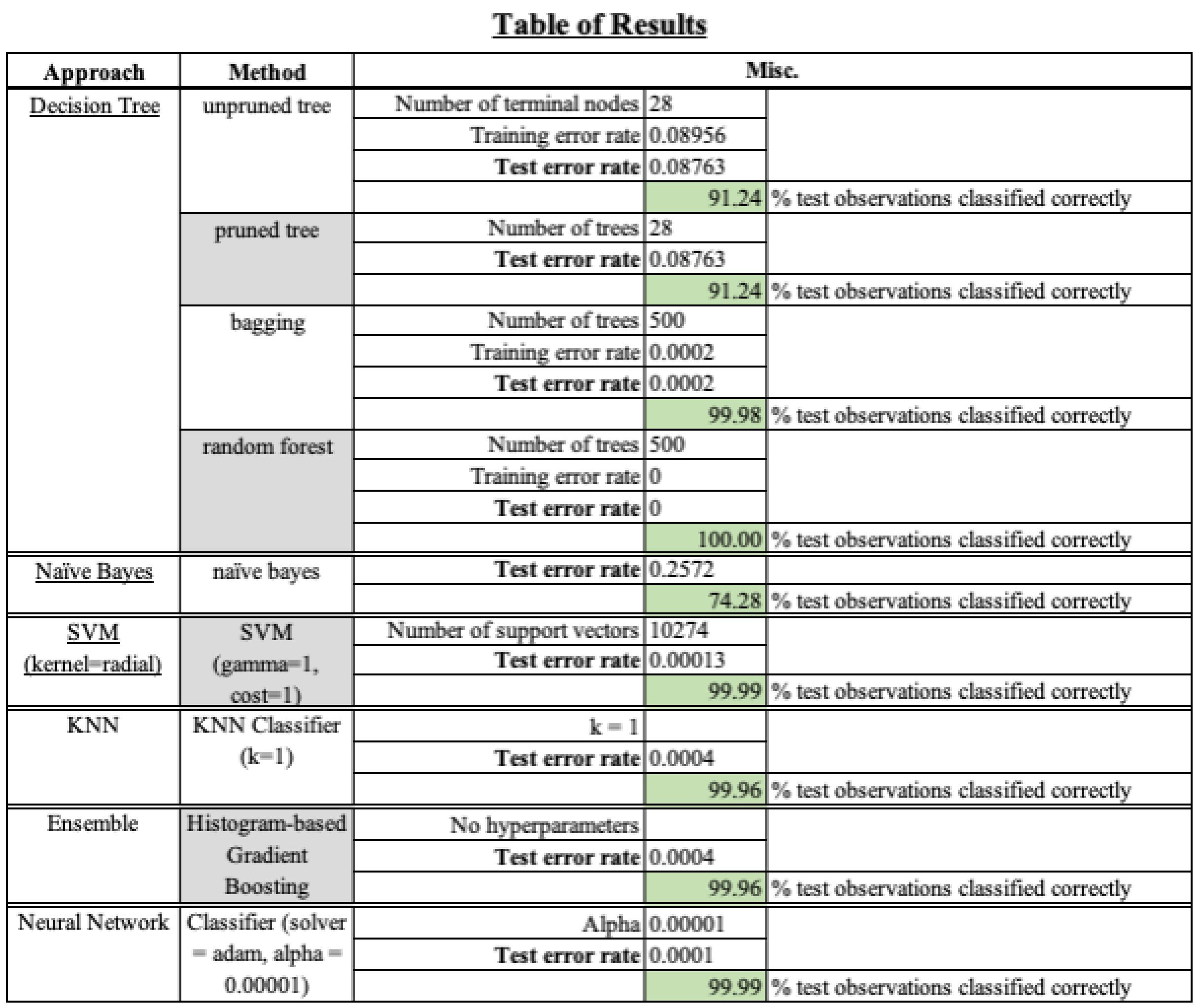


Table of Results

Approach	**Method**	**Misc.**		
Decision Tree	unpruned tree	Number of terminal nodes	28	
		Training error rate	0.08956	
		Test error rate	0.08763	
			91.24	% test observations classified correctly
	pruned tree	Number of trees	28	
		Test error rate	0.08763	
			91.24	% test observations classified correctly
	bagging	Number of trees	500	
		Training error rate	0.0002	
		Test error rate	0.0002	
			99.98	% test observations classified correctly
	random forest	Number of trees	500	
		Training error rate	0	
		Test error rate	0	
			100.00	% test observations classified correctly
Naïve Bayes	naïve bayes	**Test error rate**	0.2572	
			74.28	% test observations classified correctly
SVM (kernel=radial)	SVM (gamma=1, cost=1)	Number of support vectors	10274	
		Test error rate	0.00013	
			99.99	% test observations classified correctly
KNN	KNN Classifier (k=1)	k = 1		
		Test error rate	0.0004	
			99.96	% test observations classified correctly
Ensemble	Histogram-based Gradient Boosting	No hyperparameters		
		Test error rate	0.0004	
			99.96	% test observations classified correctly
Neural Network	Classifier (solver = adam, alpha = 0.00001)	Alpha	0.00001	
		Test error rate	0.0001	
			99.99	% test observations classified correctly

Figure D

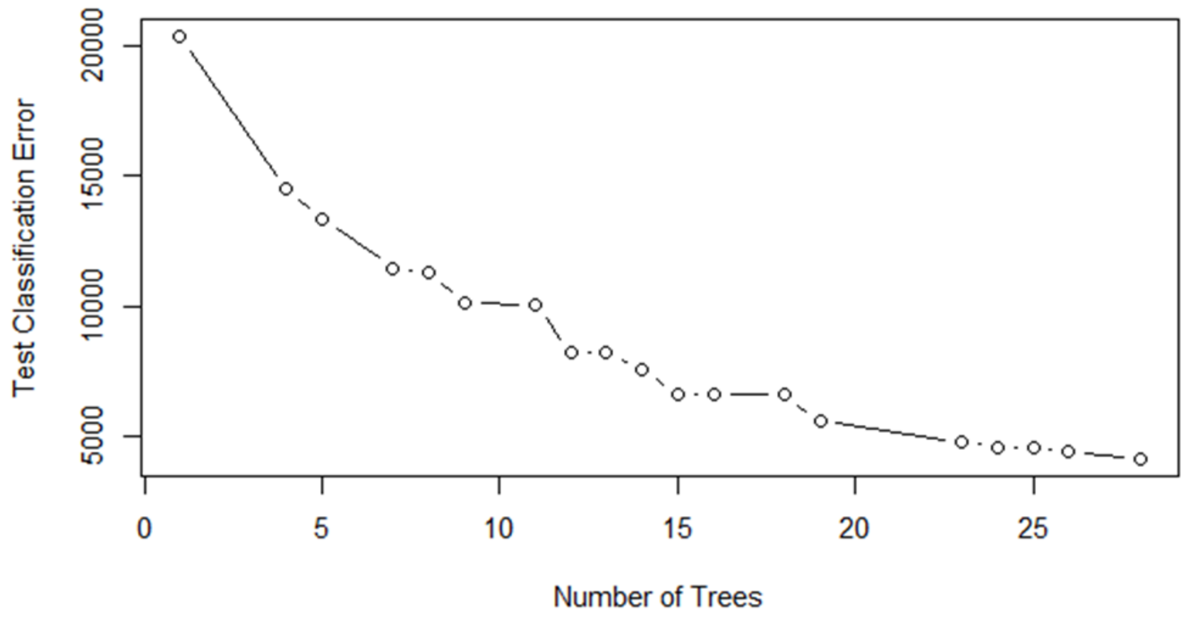
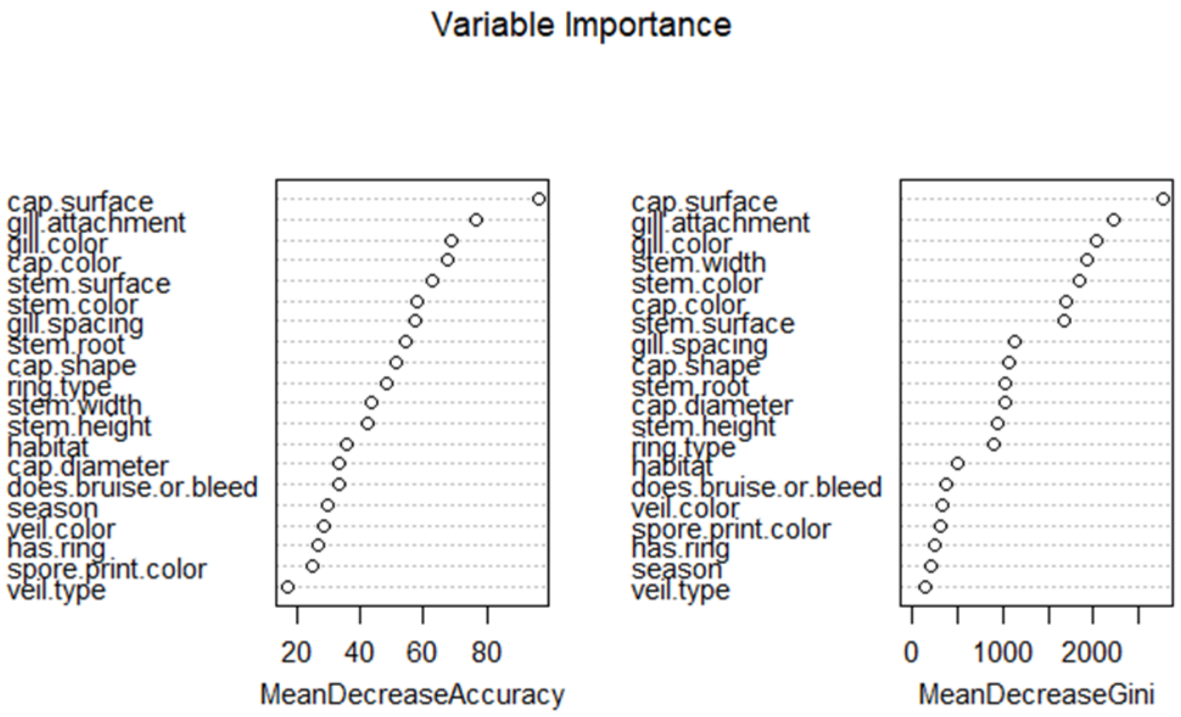


Figure E



Comparison of Approaches

The results from the various approaches used in this study are many as seen in the previous section (Results from Approaches). This following section will compare all the results from the various approaches to see which approaches performed good and performed poorly. The Naive Bayes approach was the most obvious approach used that performed poorly. This was no surprise to us because there were other approaches that we used which could handle the large amount of qualitative predictor variables that were in the mushrooms dataset. The unpruned tree and pruned tree methods (from the Decision Tree approach) performed to the same standard by having 91.24% of the test observations being correctly classified. The bagging and random forest methods performed better than both the unpruned/pruned tree and naive bayes methods. This was no surprise to us again as the bagging and random forest methods are often used to gain better prediction accuracy (and lower variance). The bagging method resulted in 99.98% of the test observations correctly classified, which was 25.7% more accurate than the Naive Bayes method and 8.7% more accurate than the unpruned/pruned tree method(s). The KNN and Histogram-based Gradient Boosting classifiers both performed at 99.96% accuracy, almost as good as the bagging method. Our best two models were the radial SVC which on some executions performed with a perfect accuracy score of 100%, as well as the Neural Network model which had a test accuracy of 99.99%.

Conclusions

What seemed like a strange and unintuitive problem to solve at first quickly became an interesting and satisfying problem to work on. As our variable relationship diagrams show, you can start to notice certain patterns very quickly among which mushrooms are safe or poisonous just with the naked eye. We tried a variety of machine learning approaches to classify mushrooms from the dataset and learn more about what can help us correctly make these classifications. While we are satisfied with our exploration of this dataset, we have been left with many interesting questions by the work done in this project. Would it be possible to identify mushrooms or plants that have specific properties, such as poisonous versus medicinal? Can we expand any of the approaches we used to classify mushrooms in this project in any other circumstances that would be useful? These are questions that someone interested in further research may find interesting to begin with.

References

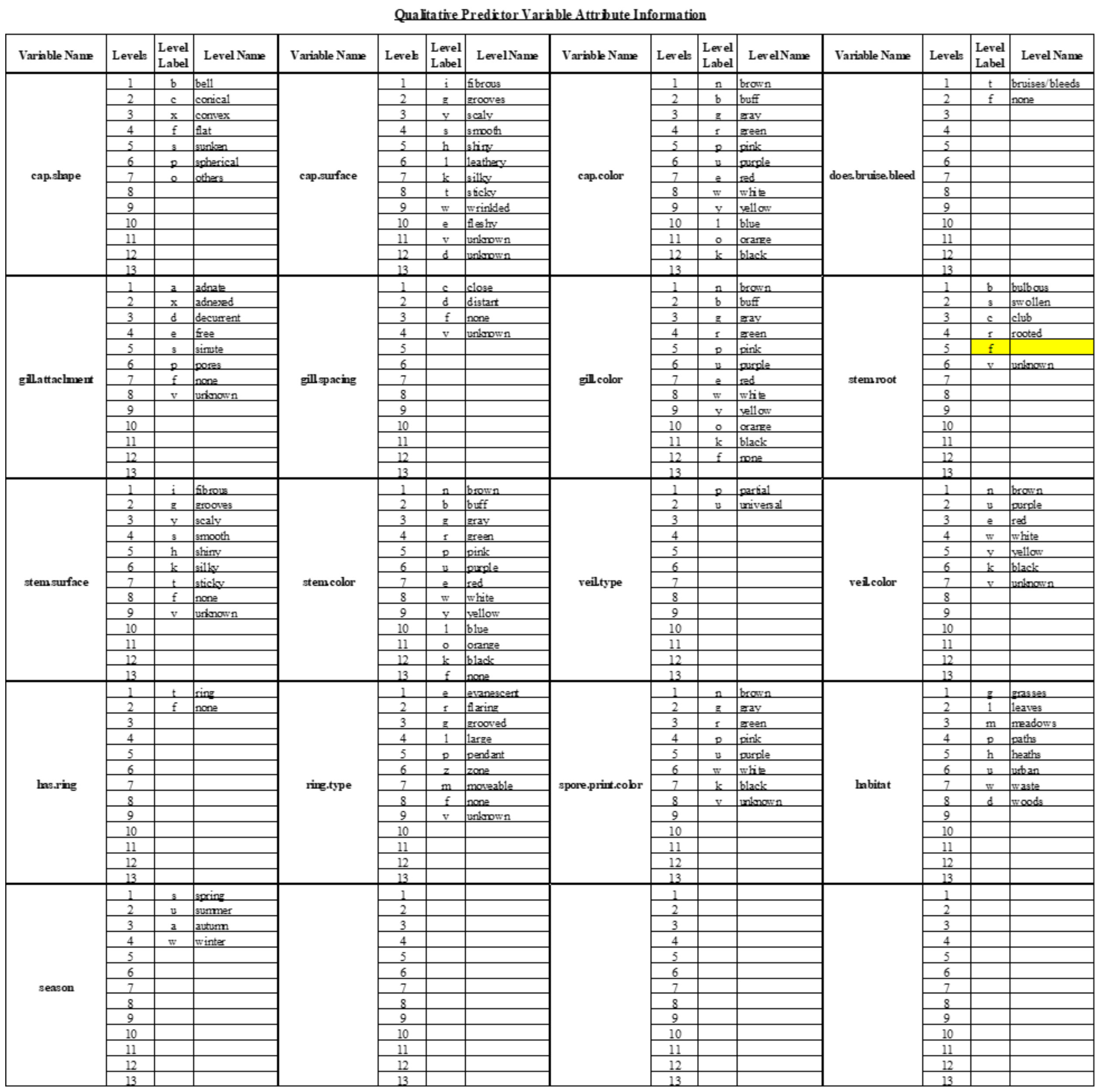
"Fungus." *Encyclopedia Britannica*. Encyclopaedia Britannica, Inc.. 21 Mar. 2023. WEB. https://www.britannica.com/science/fungus.

"Mushrooms." *The Nutrition Source*. Harvard T.H. Chan, School of Public Health. WEB. https://www.hsph.harvard.edu/nutritionsource/food-features/mushrooms/.

Wagner, D. Secondary Mushroom Dataset Data Set. (version?). UCI Machine Learning Repository. 11 Apr. 2021. WEB. https://archive.ics.uci.edu/ml/datasets/Secondary+Mushroom+Dataset

Appendix A: Supporting Images

Figure F



Qualitative Predictor Variable Attribute Information

Variable Name	Levels	Level Label	Level Name	Variable Name	Levels	Level Label	Level Name	Variable Name	Levels	Level Label	Level Name	Variable Name	Levels	Level Label	Level Name
	1	b	bell		1	i	fibrous		1	n	brown		1	t	bruises/bleeds
	2	c	conical		2	g	grooves		2	b	buff		2	f	none
	3	x	convex		3	y	scaly		3	g	gray		3		
	4	f	flat		4	s	smooth		4	r	green		4		
	5	s	sunken		5	h	shiny		5	p	pink		5		
	6	p	spherical		6	l	leathery		6	u	purple		6		
cap.shape	7	o	others	cap.surface	7	k	silky	cap.color	7	e	red	does.bruise.bleed	7		
	8				8	t	sticky		8	w	white		8		
	9				9	w	wrinkled		9	y	yellow		9		
	10				10	e	fleshy		10	l	blue		10		
	11				11	v	unknown		11	o	orange		11		
	12				12	d	unknown		12	k	black		12		
	13				13				13				13		
	1	a	adnate		1	c	close		1	n	brown		1	b	bulbous
	2	x	adnexed		2	d	distant		2	b	buff		2	s	swollen
	3	d	decurrent		3	f	none		3	g	gray		3	c	club
	4	e	free		4	v	unknown		4	r	green		4	r	rooted
	5	s	sinute		5				5	p	pink		5	f	
	6	p	pores		6				6	u	purple		6	v	unknown
gill.attachment	7	f	none	gill.spacing	7			gill.color	7	e	red	stem.root	7		
	8	v	unknown		8				8	w	white		8		
	9				9				9	y	yellow		9		
	10				10				10	o	orange		10		
	11				11				11	k	black		11		
	12				12				12	f	none		12		
	13				13				13				13		
	1	i	fibrous		1	n	brown		1	p	partial		1	n	brown
	2	g	grooves		2	b	buff		2	u	universal		2	u	purple
	3	y	scaly		3	g	gray		3				3	e	red
	4	s	smooth		4	r	green		4				4	w	white
	5	h	shiny		5	p	pink		5				5	y	yellow
	6	k	silky		6	u	purple		6				6	k	black
stem.surface	7	t	sticky	stem.color	7	e	red	veil.type	7			veil.color	7	v	unknown
	8	f	none		8	w	white		8				8		
	9	v	unknown		9	y	yellow		9				9		
	10				10	l	blue		10				10		
	11				11	o	orange		11				11		
	12				12	k	black		12				12		
	13				13	f	none		13				13		
	1	t	ring		1	e	evanescent		1	n	brown		1	g	grasses
	2	f	none		2	r	flaring		2	g	gray		2	l	leaves
	3				3	g	grooved		3	r	green		3	m	meadows
	4				4	l	large		4	p	pink		4	p	paths
	5				5	p	pendant		5	u	purple		5	h	heaths
	6				6	z	zone		6	w	white		6	u	urban
has.ring	7			ring.type	7	m	moveable	spore.print.color	7	k	black	habitat	7	w	waste
	8				8	f	none		8	v	unknown		8	d	woods
	9				9	v	unknown		9				9		
	10				10				10				10		
	11				11				11				11		
	12				12				12				12		
	13				13				13				13		
	1	s	spring		1				1				1		
	2	u	summer		2				2				2		
	3	a	autumn		3				3				3		
	4	w	winter		4				4				4		
	5				5				5				5		
	6				6				6				6		
season	7				7				7				7		
	8				8				8				8		
	9				9				9				9		
	10				10				10				10		
	11				11				11				11		
	12				12				12				12		
	13				13				13				13		

Figure G

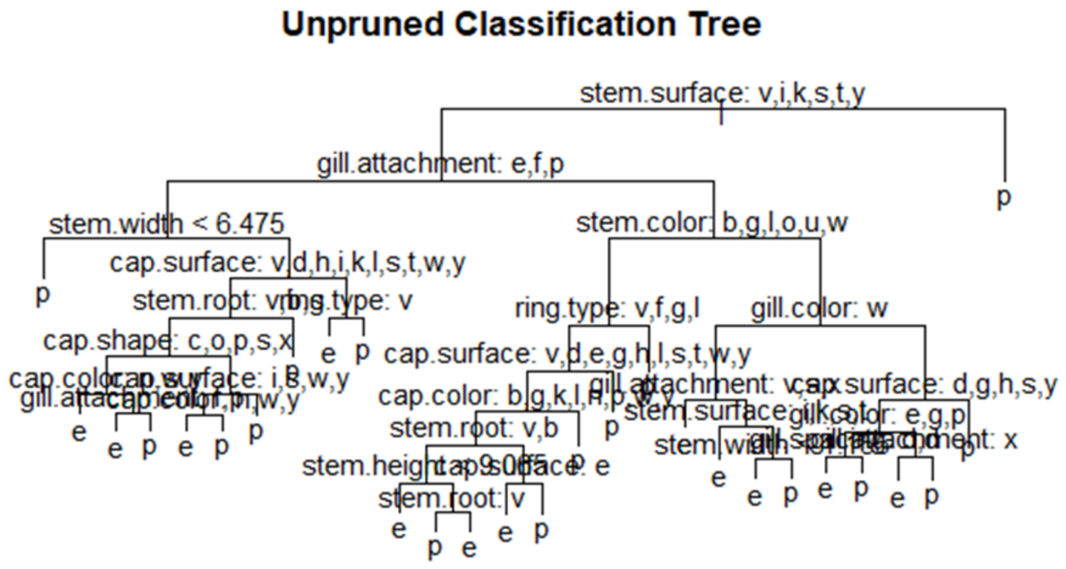


Figure H

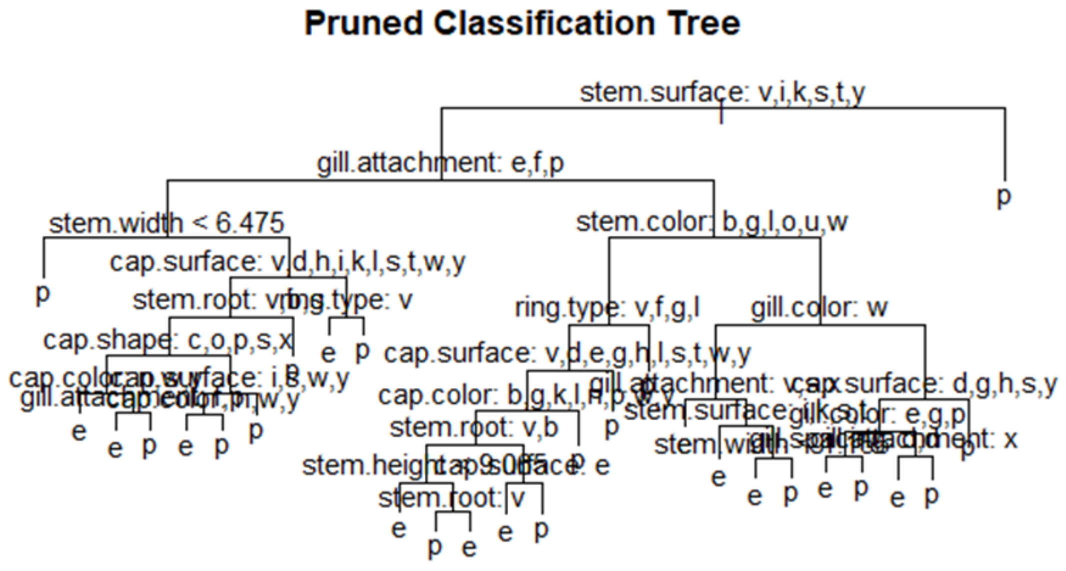
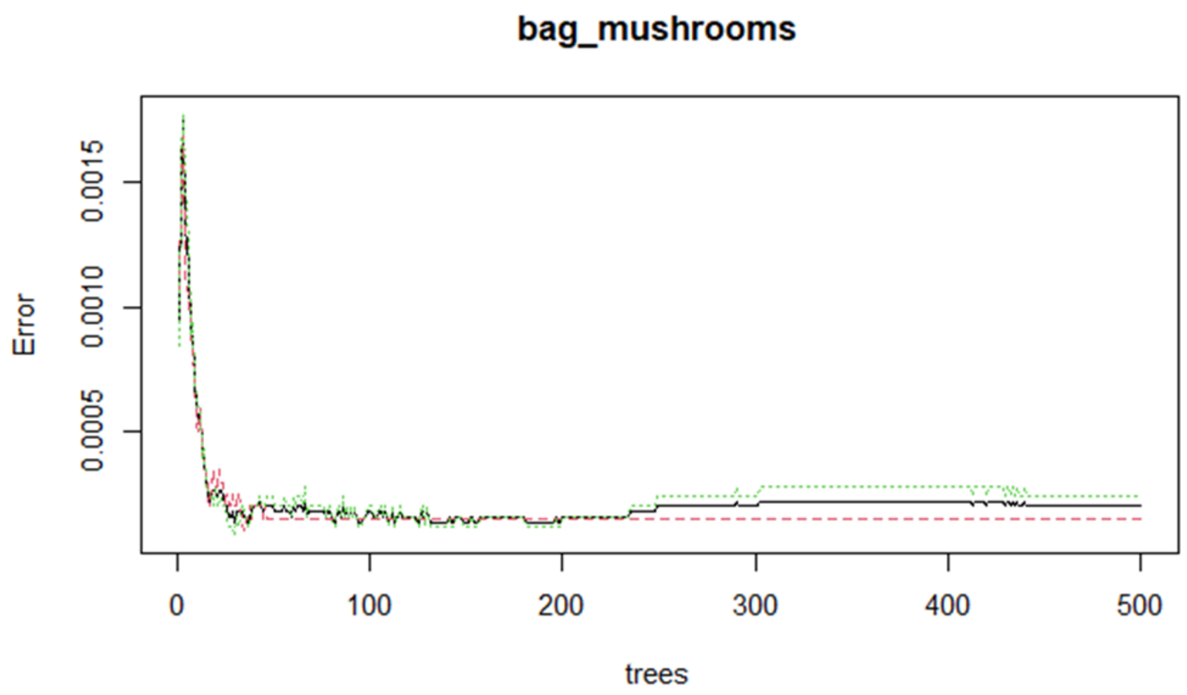


Figure I


bag_mushrooms
Error
0.0005
0.0010
0.0015
trees
0
100
200
300
400
500

Appendix B: Code

KNN

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

import warnings

warnings.filterwarnings("ignore")

df_csv = pd.read_csv('secondary_data.csv', sep=';')
features = ['cap-diameter', 'cap-shape', 'cap-surface', 'cap-color',
'does-bruise-or-bleed', 'gill-attachment',
             'gill-spacing', 'gill-color', 'stem-height', 'stem-width',
'stem-root', 'stem-surface', 'stem-color',
             'veil-type', 'veil-color', 'has-ring', 'ring-type',
'spore-print-color', 'habitat', 'season']
output = ['class']

numerical = ['cap-diameter', 'stem-height', 'stem-width']
factors = [x for x in features if x not in numerical]

df_csv.head()
df_features = df_csv[features]
df_output = df_csv[output]
df_features[factors] = df_features[factors].apply(lambda x: pd.factorize(x)[0])

imp_numerical = SimpleImputer(missing_values=np.nan, strategy="median")
imp_factor = SimpleImputer(missing_values=np.nan, strategy="most_frequent")
imp_numerical.fit(df_features[numerical])
imp_factor.fit(df_features[factors])
df_features[numerical] = imp_numerical.transform(df_features[numerical])
df_features[factors] = imp_factor.transform(df_features[factors])

X_train, X_test, y_train, y_test = train_test_split(df_features, df_output,
test_size=0.25, random_state=15)

df_xtrain = pd.DataFrame(X_train, columns=features)
df_ytrain = pd.DataFrame(y_train, columns=output)
df_xtest = pd.DataFrame(X_test, columns=features)
df_ytest = pd.DataFrame(y_test, columns=output)

for i in range(1, 20):
	neigh = KNeighborsClassifier(n_neighbors=i)
```

```
       neigh.fit(df_xtrain, df_ytrain)
       prediction = neigh.predict(df_xtest)
       acc = accuracy_score(df_ytest, prediction)
       print(i, acc)

# For mean replacement:   1 0.9996070212208541
# For median replacement: 1 0.9996070212208541
```

Histogram-based Gradient Boosting Classifier

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.metrics import accuracy_score

import warnings

warnings.filterwarnings("ignore")

df_csv = pd.read_csv('secondary_data.csv', sep=';')
features = ['cap-diameter', 'cap-shape', 'cap-surface', 'cap-color',
'does-bruise-or-bleed', 'gill-attachment',
             'gill-spacing', 'gill-color', 'stem-height', 'stem-width',
'stem-root', 'stem-surface', 'stem-color',
             'veil-type', 'veil-color', 'has-ring', 'ring-type',
'spore-print-color', 'habitat', 'season']
output = ['class']

numerical = ['cap-diameter', 'stem-height', 'stem-width']
factors = [x for x in features if x not in numerical]

df_csv.head()
df_features = df_csv[features]
df_output = df_csv[output]

df_features[factors] = df_features[factors].apply(lambda x: pd.factorize(x)[0])

X_train, X_test, y_train, y_test = train_test_split(df_features, df_output,
test_size=0.25, random_state=15)

df_xtrain = pd.DataFrame(X_train, columns=features)
df_ytrain = pd.DataFrame(y_train, columns=output)
df_xtest = pd.DataFrame(X_test, columns=features)
df_ytest = pd.DataFrame(y_test, columns=output)

# By this point the train/test splits are accessible as arrays or dataframes
```

```
clf = HistGradientBoostingClassifier()
clf.fit(df_xtrain, df_ytrain)
prediction = clf.predict(df_xtest)
acc = accuracy_score(df_ytest, prediction)
print(acc)

# 0.9996070212208541
```

Support Vector Machine (Radial kernel, gamma = 1, C = 1)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from sklearn import svm

import warnings

warnings.filterwarnings("ignore")

df_csv = pd.read_csv('secondary_data.csv', sep=';')
features = ['cap-diameter', 'cap-shape', 'cap-surface', 'cap-color',
'does-bruise-or-bleed', 'gill-attachment',
             'gill-spacing', 'gill-color', 'stem-height', 'stem-width',
'stem-root', 'stem-surface', 'stem-color',
             'veil-type', 'veil-color', 'has-ring', 'ring-type',
'spore-print-color', 'habitat', 'season']
output = ['class']

numerical = ['cap-diameter', 'stem-height', 'stem-width']
factors = [x for x in features if x not in numerical]

df_csv.head()
df_features = df_csv[features]
df_output = df_csv[output]
df_features[factors] = df_features[factors].apply(lambda x: pd.factorize(x)[0])

imp_numerical = SimpleImputer(missing_values=np.nan, strategy="mean")
imp_factor = SimpleImputer(missing_values=np.nan, strategy="most_frequent")
imp_numerical.fit(df_features[numerical])
imp_factor.fit(df_features[factors])
df_features[numerical] = imp_numerical.transform(df_features[numerical])
df_features[factors] = imp_factor.transform(df_features[factors])

X_train, X_test, y_train, y_test = train_test_split(df_features, df_output,
test_size=0.25, random_state=15)
```

```
df_xtrain = pd.DataFrame(X_train, columns=features)
df_ytrain = pd.DataFrame(y_train, columns=output)
df_xtest = pd.DataFrame(X_test, columns=features)
df_ytest = pd.DataFrame(y_test, columns=output)

print("Working")

clf = svm.SVC(kernel='rbf', gamma=1, C=1)
clf.fit(df_xtrain, df_ytrain)
prediction = clf.predict(df_xtest)
acc = accuracy_score(df_ytest, prediction)
print(acc)

#0.9818574796960964
```

Neural Network (‘adam’ solver, alpha = 0.00001)

```
from sklearn.neural_network import MLPClassifier
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

import warnings

warnings.filterwarnings("ignore")

df_csv = pd.read_csv('secondary_data.csv', sep=';')
features = ['cap-diameter', 'cap-shape', 'cap-surface', 'cap-color',
'does-bruise-or-bleed', 'gill-attachment',
             'gill-spacing', 'gill-color', 'stem-height', 'stem-width',
'stem-root', 'stem-surface', 'stem-color',
             'veil-type', 'veil-color', 'has-ring', 'ring-type',
'spore-print-color', 'habitat', 'season']
output = ['class']

numerical = ['cap-diameter', 'stem-height', 'stem-width']
factors = [x for x in features if x not in numerical]

df_features = df_csv[features]
df_output = df_csv[output]

df_features[factors] = df_features[factors].apply(lambda x: pd.factorize(x)[0])

X_train, X_test, y_train, y_test = train_test_split(df_features, df_output,
test_size=0.25, random_state=15)
```

```
df_xtrain = pd.DataFrame(X_train, columns=features)
df_ytrain = pd.DataFrame(y_train, columns=output)
df_xtest = pd.DataFrame(X_test, columns=features)
df_ytest = pd.DataFrame(y_test, columns=output)

clf = MLPClassifier(solver='adam', alpha=1e-5, random_state=1)
clf.fit(df_xtrain, df_ytrain)
prediction = clf.predict(df_xtest)
acc = accuracy_score(df_ytest, prediction)
print(acc)

# 0.999934503536809
```

Load and Clean Data

```
# libraries
library(ggplot2)
library(class) #for KNN
library(e1071) #for Naive Bayes, SVM
library(leaps) #for Regularization, regsubsets
library(tree)
library(randomForest)

# read data
mushrooms <- read.csv(file = 'secondary_data2.csv',stringsAsFactors = TRUE)

#check for NAs
sum(is.na(mushrooms))

#edit 1
levels(mushrooms$cap.surface)[1] <- "v"

#edit 2
levels(mushrooms$gill.attachment)[1] <- "v"

#edit 3
levels(mushrooms$gill.spacing)[1] <- "v"

#edit 4
levels(mushrooms$stem.root)[1] <- "v"

#edit 5
levels(mushrooms$stem.surface)[1] <- "v"

#edit 6
levels(mushrooms$veil.type)[1] <- "p"
```

```
#edit 7
levels(mushrooms$veil.color)[1] <- "v"

#edit 8
levels(mushrooms$ring.type)[1] <- "v"

#edit 9
levels(mushrooms$spore.print.color)[1] <- "v"
# Summary Statistics

summary(mushrooms)


## quantiative vars.

pairs(mushrooms[,c(1,2,10,11)])

round(cor(mushrooms[,c(2,10,11)]),3)
```

Charts

```
gplot(mushrooms,aes(x=cap.surface,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Cap Surface vs. Cap Shape") +
  xlab("Cap Surface Type") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("unknown","unknown","fleshy","grooves","shiny","fibrous",
"silky","leathery","smooth","sticky","wrinkled","scaly")) +
  theme(axis.text.x = element_text(angle = 90,hjust=1,vjust=0))


ggplot(mushrooms,aes(x=cap.color,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Cap Color vs. Cap Shape") +
  xlab("Cap Color") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +
scale_x_discrete(labels=c("buff","red","gray","black","blue","brown","orange","pink
","green","purple","white","yellow")) +
  theme(axis.text.x = element_text(angle = 90,hjust=1,vjust=0))


ggplot(mushrooms,aes(x=does.bruise.or.bleed,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Does Bruise/Bleed vs. Cap Shape") +
  xlab("Does Bruise or Bleed") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +
  scale_x_discrete(labels=c("no","yes"))


ggplot(mushrooms,aes(x=gill.attachment,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Gill Attachment vs. Cap Shape") +
  xlab("Gill Attachment Type") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("unknown","adnate","decurrent","free","none","pores","sin
ute","adnexed")) +
  theme(axis.text.x = element_text(angle = 90,hjust=1,vjust=0))


ggplot(mushrooms,aes(x=gill.spacing,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Gill Spacing vs. Cap Shape") +
  xlab("Gill Spacing") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +
  scale_x_discrete(labels=c("unknown","close","distant","none"))
```

```
ggplot(mushrooms,aes(x=gill.color,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Gill Color vs. Cap Shape") +
  xlab("Gill Color") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("buff","red","none","gray","black","brown","orange","pink
","green","purple","white","yellow")) +
  theme(axis.text.x = element_text(angle = 90,hjust=1,vjust=0))


ggplot(mushrooms,aes(x=stem.root,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Stem Root vs. Cap Shape") +
  xlab("Stem Root Type") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("unknown","bulbous","club","unknown","rooted","swollen"))


ggplot(mushrooms,aes(x=stem.surface,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Stem Surface vs. Cap Shape") +
  xlab("Stem Surface Type") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("unknown","none","grooves","shiny","fibrous","silky","smo
oth","sticky","scaly")) +
  theme(axis.text.x = element_text(angle = 90,hjust=1,vjust=0))
```

```
ggplot(mushrooms,aes(x=stem.color,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Stem Color vs. Cap Shape") +
  xlab("Stem Color") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("buff","red","none","gray","black","blue","brown","orange
","pink","green","purple","white","yellow")) +
  theme(axis.text.x = element_text(angle = 90,hjust=1,vjust=0))


ggplot(mushrooms,aes(x=veil.type,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Veil Type vs. Cap Shape") +
  xlab("Veil Type") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +
  scale_x_discrete(labels=c("partial","universal"))


ggplot(mushrooms,aes(x=veil.color,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Veil Color vs. Cap Shape") +
  xlab("Veil Color") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("unknown","red","black","brown","purple","white","yellow"
))
  # theme(axis.text.x = element_text(angle = 90,hjust=1,vjust=0))


ggplot(mushrooms,aes(x=has.ring,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Has Ring vs. Cap Shape") +
  xlab("Has Ring") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +
  scale_x_discrete(labels=c("no","yes"))


ggplot(mushrooms,aes(x=ring.type,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Ring Type vs. Cap Shape") +
  xlab("Ring Type") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("unknown","evanescent","none","grooved","large","moveable
","pendant","flaring","zone")) +
  theme(axis.text.x = element_text(angle = 90,hjust=1,vjust=0))


ggplot(mushrooms,aes(x=spore.print.color,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Spore Print Color vs. Cap Shape") +
  xlab("Spore Print Color") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("unknown","gray","black","brown","pink","green","purple",
"white"))
  # theme(axis.text.x = element_text(angle = 90,hjust=1,vjust=0))



ggplot(mushrooms,aes(x=habitat,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Habitat vs. Cap Shape") +
  xlab("Habitat") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +

scale_x_discrete(labels=c("woods","grasses","heaths","leaves","meadows","paths","ur
ban","waste"))



ggplot(mushrooms,aes(x=season,y=cap.shape,col=class)) +
  geom_jitter() +
  ggtitle("Season vs. Cap Shape") +
  xlab("Seasons") +
  ylab("Cap Shape Type") +
  theme(plot.title = element_text(face="bold")) +
  scale_color_discrete(labels=c('Edible','Poisonous')) +

scale_y_discrete(labels=c("bell","conical","flat","others","spherical","sunken","co
nvex")) +
  scale_x_discrete(labels=c("autumn","spring","summer","winter"))
```

Training/Testing Split

```
set.seed(12)

train <- sample(1:nrow(mushrooms),0.75*nrow(mushrooms))


mushrooms_train <- mushrooms[train,]
mushrooms_test <- mushrooms[-train,]

nrow(mushrooms_train)+nrow(mushrooms_test)
```

Unpruned Tree

```
# fitting classification trees
tree_mushrooms <- tree(class~.,data = mushrooms_train)

summary(tree_mushrooms)

# number of terminal nodes = 28
# training error rate = 0.08956
```

```
plot(tree_mushrooms)
text(tree_mushrooms,pretty=0)
title(main="Unpruned Classification Tree")

tree_mushrooms

tree_pred <- predict(tree_mushrooms,mushrooms_test,type="class")

table(tree_pred,mushrooms_test$class)

#correct predictions in the test data set
(6308+7622)/nrow(mushrooms_test)

#wrong pred.
1-((6308+7622)/nrow(mushrooms_test))

# 91.24% of the test observations are correctly classified (test error rate =
0.08763)
```

Pruned Tree

```
cv_mushrooms <- cv.tree(tree_mushrooms, FUN=prune.misclass)
cv_mushrooms

plot(cv_mushrooms$size,cv_mushrooms$dev,type="b",xlab = "Number of
Trees",ylab="Test Classification Error")

prune_mushrooms_misclass <- prune.misclass(tree_mushrooms,best=28)
plot(prune_mushrooms_misclass)
text(prune_mushrooms_misclass,pretty=0)
title(main="Pruned Classification Tree")

tree_pred2 <- predict(prune_mushrooms_misclass,mushrooms_test,type="class")
table(tree_pred2,mushrooms_test$class)

#the test observations that are correctly classified
(6308+7622)/nrow(mushrooms_test)

#test error rate
1-((6308+7622)/nrow(mushrooms_test))

# 91.24% of the test observations are correctly classified (test error rate =
0.08763)
```

Random Forest

```
### bagging
```

```
set.seed(54)

bag_mushrooms <- randomForest(class~.,data =
mushrooms_train,mtry=20,importance=TRUE) #mtry=20 indicates all 20 predictors
should be considered
Bag_mushrooms

(20353+25439)/nrow(mushrooms_train)

#training error rate
1-((20353+25439)/nrow(mushrooms_train))

# 99.98% of the training observations are correctly classified (training error rate
= 0.0002)

plot(bag_mushrooms)

tree_pred3 <- predict(bag_mushrooms,mushrooms_test,type="class")
table(tree_pred3,mushrooms_test$class)

(6825+8440)/nrow(mushrooms_test)

#test error rate
1-((6825+8440)/nrow(mushrooms_test))

# 99.98% of the test observations are correctly classified (test error rate =
0.0002)

set.seed(73)

bag_mushrooms <- randomForest(class~.,data = mushrooms_train,mtry=20,ntree=28) #28
number of trees
bag_mushrooms

plot(bag_mushrooms)

tree_pred4 <- predict(bag_mushrooms,mushrooms_train,type="class")
table(tree_pred4,mushrooms_train$class)

(20356+25445)/nrow(mushrooms_train)

set.seed(28)

m <- round(sqrt(20),0)

rf_mushrooms <- randomForest(class~.,data = mushrooms_train,mtry=m,importance=TRUE)
```

```
rf_mushrooms

# training error rate = 0



tree_pred5 <- predict(rf_mushrooms,mushrooms_test,type="class")
table(tree_pred5,mushrooms_test$class)

(6825+8443)/nrow(mushrooms_test)

importance(rf_mushrooms)
varImpPlot(rf_mushrooms,main="Variable Importance")
```

Naive Bayes

```
nb_fit <- naiveBayes(class~.,data = mushrooms_train)
# nb_fit

nb_class <- predict(nb_fit,mushrooms_test)
table(nb_class,mushrooms_test$class)

mean(nb_class==mushrooms_test$class)

#test error rate
1-(mean(nb_class==mushrooms_test$class))

# 74.28% of test observations are correctly classified (test error rate = 0.25720)
```

Support Vector Machine

```
svm_fit <- svm(class~.,data = mushrooms_train,kernel="radial",gamma=1,cost=1)

summary(svm_fit)

# number of support vectors = 10274
# (5557,4747)

tree_pred6 <- predict(svm_fit,mushrooms_test,type="class")
table(tree_pred6,mushrooms_test$class)

(6823+8443)/nrow(mushrooms_test)

#test error rate
1-((6823+8443)/nrow(mushrooms_test))

# 99.99% test observations are correctly classified (test error rate = 0.00013)
```

```
## tune out

#NOTICE/CAUTION: this will RUN for APPROX. 2 HOURS 15 MINS plus (depending on PC)

tune_out1 <- tune(svm,class~.,data =
mushrooms_train,kernel="radial",cost=c(0.01,0.1,1,5,10))

summary(tune_out1)
tune_out1$best.performance

#NOTICE/CAUTION: this will RUN for APPROX. 4 HOURS 30 MINS plus (depending on PC)

tune_out2 <- tune(svm,class~.,data = mushrooms_train,kernel="radial",ranges =
list(cost=c(0.01,0.1,1,5),gamma=1))

summary(tune_out2)

table(true=mushrooms_test$class,pred=predict(tune_out2$best.model,newdata =
mushrooms_test))

(6823+8443)/nrow(mushrooms_test)

#test error rate
1-((6823+8443)/nrow(mushrooms_test))

# test error rate = 0.00013
```