HarvardX; Data Science Capstone

Choose Your Own Project

Mushroom Classification

Tali Behar

December 2020

1 Executive Summary

1.1 Overview

During Covid-19 time, I started taking daily walks around the neighberhood, usually in the same path. I then started noticing an interesting sight, which is hunderds of mushrooms growing everytime after it rains (and I live in a very rainy area). They had different shapes, colors, texture and size, depending on where they grew. I read a bit in wikipedia about different kinds of mushrooms, in the hope that there is an easy way to determine whether a mushroom is edible (or even touchable), but it is far from being the case. When I was searching for a topic for the CYO project, I came accross a huge dataset about mushrooms, and couldn't resist the opportunity to analyze it.

1.2 Background

Our dataset is taken from: https://www.kaggle.com/uciml/mushroom-classification

This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. The latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy.

The data is categorical - each column contain its own factors, represented by unique letters (see extension on section 2.3)

1.3 Goal

1. Determine which features are most indicative of a poisonous mushroom.
2. Determine what types of ML models perform best on 'Mushroom Classification' dataset by achieving model accuracy of 100%, in a reasonable runtime.

1.4 Variable Selection

1. **Y = class**

The outcome will be the class of the mushroom: e = edible, p = poisonous

2. **The features**

For classifying mushrooms, whether it's outside or through images, it's important to be familiar with all the details about them. I chose to take all of the 22 mushrooms characteristics to be used for the prediction: Odor, population, habitat, bruises, cap, gill, stalk, veil, ring and spore-print-color.

1.5 Method

The ML algorithms that were chosen to model the mushroom classification dataset come from the family of **Classification Algorithms** and are: Random forest, Classification trees, and K-nearest neighbors (KNN).

- Divide the mushrooms dataset randomly to train set and test set.
- Visualize the distribution of the characteristics between e/p class and find meaningful insights.
- Visualize the relationships and correlation in the data - once between the characteristics and the class, and once among the characteristics themselves.
- Train and fit the above algorithms using cross-validation and tuning methods.
- Omit the unimportant features.
- Construct canonical dataset for applying the KNN algorithm.

1.6 Model Estimation

Evaluate model performance based on its **overall accuracy** and runtime.

Overall accuracy is the simplest way to evaluate the algorithm when the outcomes are categorical. This method is simply reporting the proportion of cases that were correctly predicted in the test set.

1.7 Model Results

Random forest, KNN and Classification trees (model 1.2) achieved the perfect 100% accuracy.

1.8 Conclusion

- The chosen models for predicting the mushrooms class are Random Forest and KNN, which are more flexible and easier to train and tune, and are better for classifying.
- Random forest performed better runtime- 422% faster than KNN runtime.
- The most important features for the classification are 'spore.print.color', 'gill.color', 'stalk.surface.above.ring' and 'stalk.surface.below.ring'

2 Data Preparation

2.1 Installing Required Packages

```
library(readr)
library(tidyverse)
library(caret)
library(gridExtra)
library(knitr)
library(plot.matrix)
library(rpart)
library(rpart.plot)
library(ggplot2)
library(randomForest)
```

2.2 Loading Mushrooms Classification Dataset

Note: this project was developed on R version 3.6.1 and might not be compatible with newer versions

```
# Importing the data - read csv file locally
# Note: original dataset can be found here: https://www.kaggle.com/uciml/mushroom-classification

mushrooms_df <- read.csv("mushrooms.csv")
mushrooms <- mushrooms_df
```

2.3 Get A Glimpse Of Mushrooms Classification Dataset

class	nrow	ncol
data.frame	8124	23

```
## Rows: 8,124
## Columns: 23
## $ class                    <fct> p, e, e, p, e, e, e, e, p, e, e, e, e, p, ...
## $ cap.shape                <fct> x, x, b, x, x, x, b, b, x, b, x, x, b, x, ...
## $ cap.surface              <fct> s, s, s, y, s, y, s, y, y, s, y, y, s, y, ...
## $ cap.color                <fct> n, y, w, w, g, y, w, w, w, y, y, y, y, w, ...
## $ bruises                  <fct> t, t, t, t, f, t, t, t, t, t, t, t, t, t, ...
## $ odor                     <fct> p, a, l, p, n, a, a, l, p, a, l, a, a, p, ...
## $ gill.attachment          <fct> f, f, f, f, f, f, f, f, f, f, f, f, f, f, ...
## $ gill.spacing             <fct> c, c, c, c, w, c, c, c, c, c, c, c, c, c, ...
## $ gill.size                <fct> n, b, b, n, b, b, b, b, n, b, b, b, b, n, ...
## $ gill.color               <fct> k, k, n, n, k, n, g, n, p, g, g, n, w, k, ...
## $ stalk.shape              <fct> e, e, e, e, t, e, e, e, e, e, e, e, e, e, ...
## $ stalk.root               <fct> e, c, c, e, e, c, c, c, e, c, c, c, c, e, ...
## $ stalk.surface.above.ring <fct> s, s, s, s, s, s, s, s, s, s, s, s, s, s, ...
## $ stalk.surface.below.ring <fct> s, s, s, s, s, s, s, s, s, s, s, s, s, s, ...
## $ stalk.color.above.ring   <fct> w, w, w, w, w, w, w, w, w, w, w, w, w, w, ...
## $ stalk.color.below.ring   <fct> w, w, w, w, w, w, w, w, w, w, w, w, w, w, ...
```

```
## $ veil.type                <fct> p, p, p, p, p, p, p, p, p, p, p, p, p, p, p, ...
## $ veil.color               <fct> w, w, w, w, w, w, w, w, w, w, w, w, w, w, w, ...
## $ ring.number              <fct> o, o, o, o, o, o, o, o, o, o, o, o, o, o, o, ...
## $ ring.type                <fct> p, p, p, p, e, p, p, p, p, p, p, p, p, p, p, ...
## $ spore.print.color        <fct> k, n, n, k, n, k, k, n, k, k, n, k, n, n, ...
## $ population               <fct> s, n, n, s, a, n, n, s, v, s, n, s, s, v, ...
## $ habitat                  <fct> u, g, m, u, g, g, m, m, g, m, g, m, g, u, ...
```

2.3.1 The mushrooms datadet contains 8k rows and 23 columns.

columns

The first column represents the class of the mushroom, "e" for edible and "P" for poisonous.

Columns 2:23 represents the mushrooms characteristics as the following:

Odor, Population, Habitat, Bruises, Cap, Gill, Stalk, Veil, Ring and Spore-print-color.

We can distinguish between visual and general characteristics:

- Visual characteristics can be measured by sight (images), such as the color and shapes of the mushroom's cap, bruises, gill, stalk, veil and ring.
- General characteristics must be evaluated in-person, such as the mushroom's odor, the habitat, population and Spore-print-color.

Rows

Each row represents one observation on one mushroom and contain its class and unique characteristics.

Moving forward, we'll learn about the mushroom's characteristics and their influence on the mushroom's classification.

2.3.2 The dataset is categorical

The next table specifies the class of the outcome and features (factor), and the levels of each of the factors.

'veil.type' has only 1 level which is not contributing information as a feature. We'll remove it from the data and won't use it as a feature.

Some observations contain '?' in 'stalk.root' column, and we chose to use it as a regular factor, as the distribution is equal between edible and poisonous, and if we omitted these rows, we would lose 40% of the data.

mushrooms characteristics	class	levels
class	factor	2
cap.shape	factor	6
cap.surface	factor	4
cap.color	factor	10
bruises	factor	2
odor	factor	9
gill.attachment	factor	2
gill.spacing	factor	2
gill.size	factor	2
gill.color	factor	12
stalk.shape	factor	2

mushrooms characteristics	class	levels
stalk.root	factor	5
stalk.surface.above.ring	factor	4
stalk.surface.below.ring	factor	4
stalk.color.above.ring	factor	9
stalk.color.below.ring	factor	9
veil.type	factor	1
veil.color	factor	4
ring.number	factor	3
ring.type	factor	5
spore.print.color	factor	9
population	factor	6
habitat	factor	7

2.3.3 The prevalence of the observation

The prevalence of the mushrooms class in the dataset is almost about the same.

52% of the observations belong to edible type and 48% are poisonous, which means we should not expect biased prediction.

Predicting poisonous mushrooms as edible can be a terrible mistake, so in addition to the accuracy of the model, we'll also look at the other parts of the 'confusionMatrix', that contains the sensitivity, specificity, prevalence and balanced accuracy.

Mushroom class	Count	Proportion (%)
e	4208	51.79714
p	3916	48.20286

Here is a quick summary of the variables and their count.

Since the full table is very wide, we'll present only the first 5 columns

```
##  class    cap.shape cap.surface   cap.color    bruises
##  e:4208   b: 452    f:2320      n      :2284   f:4748
##  p:3916   c:   4    g:   4      g      :1840   t:3376
##           f:3152    s:2556      e      :1500
##           k: 828    y:3244      y      :1072
##           s:  32                w      :1040
##           x:3656                b      : 168
##                                 (Other): 220
```

2.4 Create training and test sets

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`

# Test set will be 20% of mushrooms data
test_index <- createDataPartition(y = mushrooms$class,
                                  times = 1, p = 0.2, list = FALSE)
```

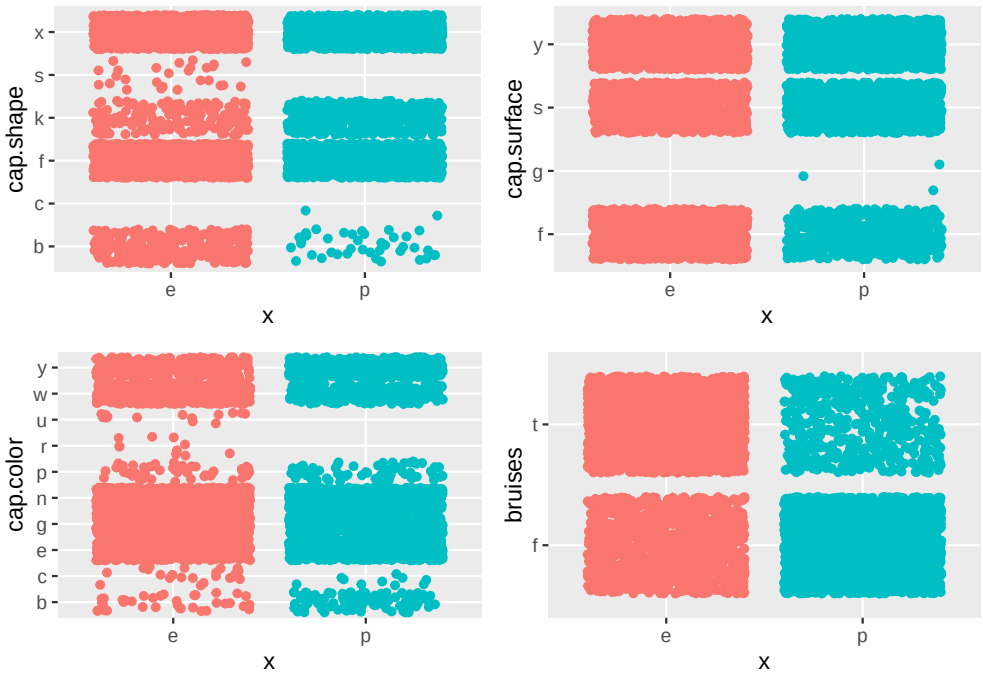
```r
train_set <- mushrooms %>% slice(-test_index)
test_set <- mushrooms %>% slice(test_index)
# Remove test_index
rm(test_index)

# Define the outcome (class of the mushrooms) = y
y <- train_set$class

# Restore the true y from test set
y_test <- test_set$class

# Remove the outcome from training and test set
test_set <- test_set %>% select(-class)
train_set <- train_set %>% select(-class)
```

3 Mushrooms classification Dataset Overview

3.1 Visual Characteristics

3.1.1 cap and bruises characteristics

- Cap
  - Differently shaped and colored upper part of the mushroom that protects the gills. In other words, the cap is the top of the mushroom.
  - Cap shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
  - Cap surface: fibrous=f, grooves=g, scaly=y, smooth=s
  - Cap color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
- Bruises
  - Mushroom bruising involves nicking the top and bottom of the mushroom cap and observing any color changes.
  - Bruises: bruises=t, no=f
- We can see that there is no significant difference in the cap characteristics or bruises that indicate class. Yet, sunken shape, purple color and green color will always indicate edible mushroom.

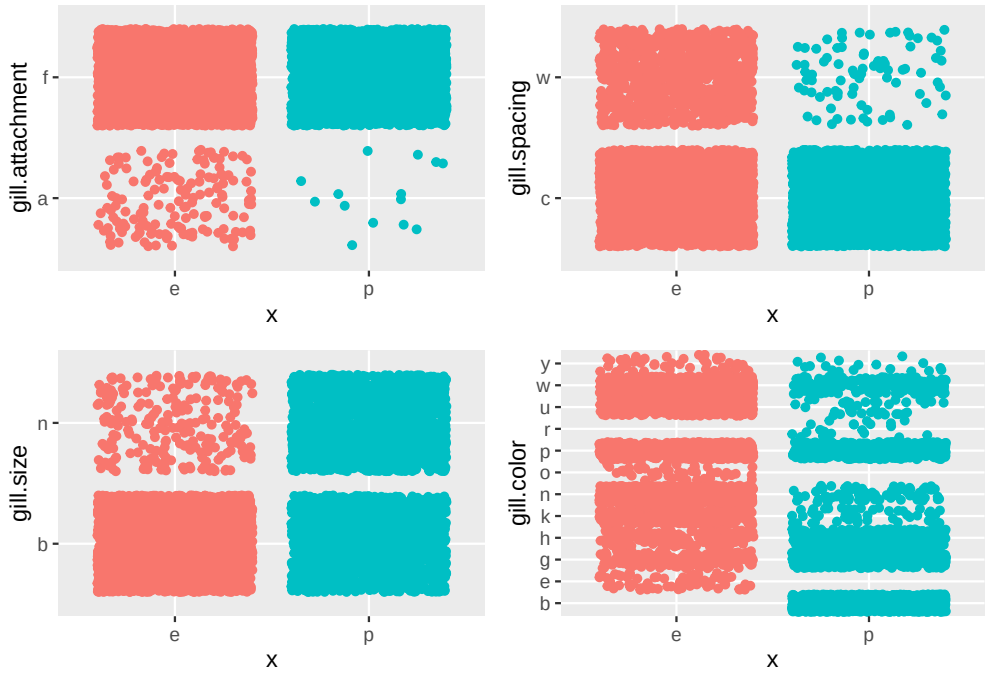


3.1.2 gill characteristics

- Fertile spore-producing part of the mushroom, located under the cap.
  - Gill attachment: attached=a, descending=d, free=f, notched=n

  - Gill spacing: close=c, crowded=w, distant=d
  - Gill size: broad=b, narrow=n
  - Gill color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y

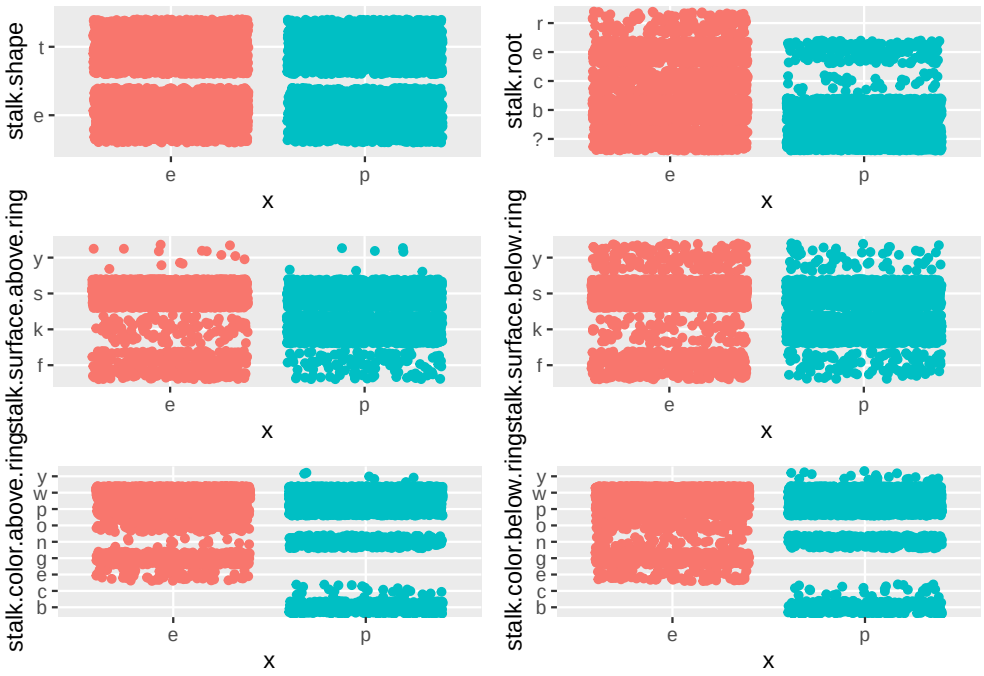
- We can see differences in the gill characteristics that may indicate the mushroom's class, especially the color. Buff and Green gill colors will always indicate poisonous.



3.1.3 Stalk characteristics

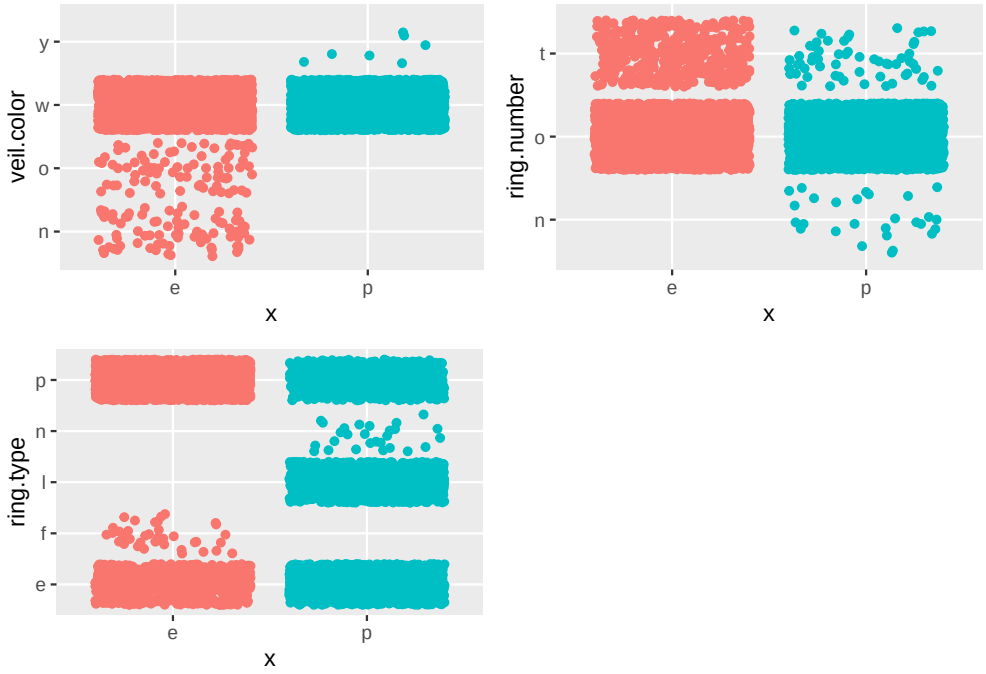
- Stem, Axis supporting the mushroom's cap. The role of the mushroom stalk is to raise the cap above the grass, twigs or stones that are close to the ground.
  - Stalk shape: enlarging=e, tapering=t
  - Stalk root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
  - Stalk surface above ring: fibrous=f, scaly=y, silky=k, smooth=s
  - Stalk surface below ring: fibrous=f, scaly=y, silky=k, smooth=s
  - Stalk color above ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
  - Stalk color below ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

- Buff, cinnamon, and yellow stalk color above ring will always indicate poisonous mushroom. All other stalk characteristics distributing in different frequency between the mushrooms class.



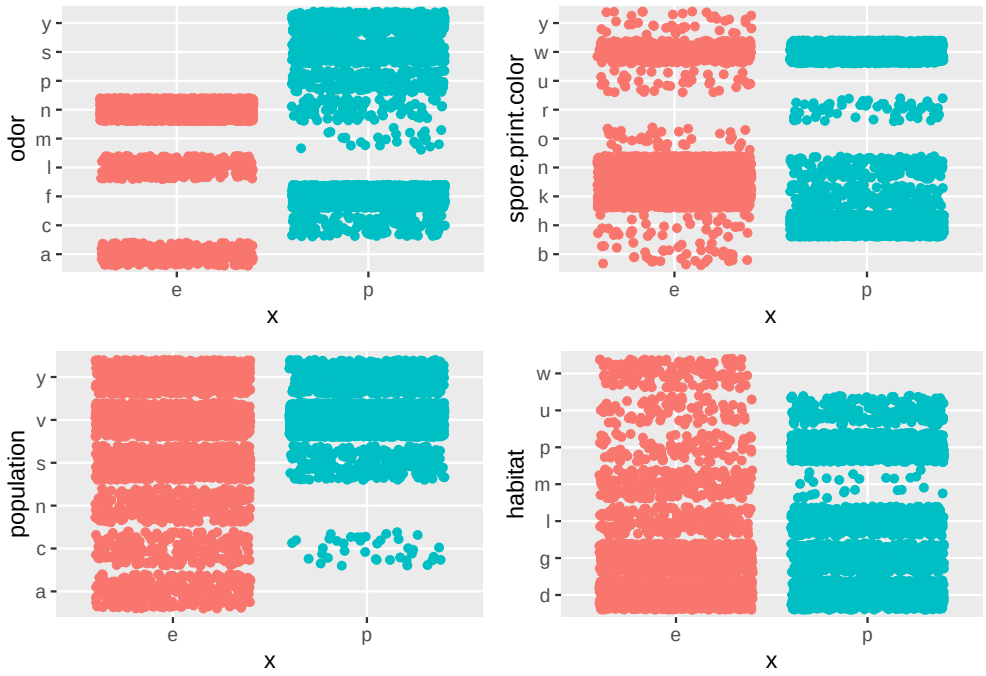
3.1.4 veil and Ring characteristics

- Veil
  - The thin membrane that covers the cap and stalk of an immature mushroom
  - Veil color: brown=n, orange=o, white=w, yellow=y
- Ring
  - Membrane located under the cap and circling the stem; remnant of a membrane that covered the gills of the immature mushroom and ruptured as the cap grew.
  - Ring number: none=n, one=o, two=t
  - Ring type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
- Brown and orange veil color will always indicate edible mushroom.
- Mushrooms with large ring or no rings will always indicate poison.



3.2 General Characteristics

- Odor - The smell of the mushroom
  - almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
  - There are significant differences in the odor distribution. Poisonous and edible mushrooms smelles very differently and we can consider the odor as a dominant feature.
- Spore print color
  - The spore print is the powdery deposit obtained by allowing spores of a fungal fruit body to fall onto a surface underneath
  - black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
  - Green print color will always indicate poisonous.
- Population - the way they spread in their habitat
  - abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
  - Poisonous mushrooms can never be found abundant or in numerous numbers.
- Habitat
  - grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d
  - Poisonous mushrooms can grow anywhere but can never be found on waste.



3.3 Relationships and correlation in the data

All the above characteristics are combining one mushroom. Since that, it can be assumed that all the features are correlated with the class of the mushroom and among themselves.

In the above plots we noticed high influence of the odor on the mushroom class. Yet, in order to select the best other features that have a strong predictive power, we need to see if they are correlated or not.

For building efficient predictive models, we would ideally only include variables that uniquely explain some amount of variance in the outcome.

3.3.1 Chi-Square Test

The **Chi-Square Test** (χ^2 test) of Independence determines whether there is an association between **categorical variables**.

It compares two variables in a contingency table to see if they are related. In a more general sense, it tests to see whether distributions of categorical variables differ from each other.

It is a nonparametric test.

A chi square test will produce a p-value which will tell us if the test results are significant or not.

Under the null hypothesis H_0 - the categorical variables are independent.

If $p.value <= \alpha$: significant result, reject H_0, dependent.

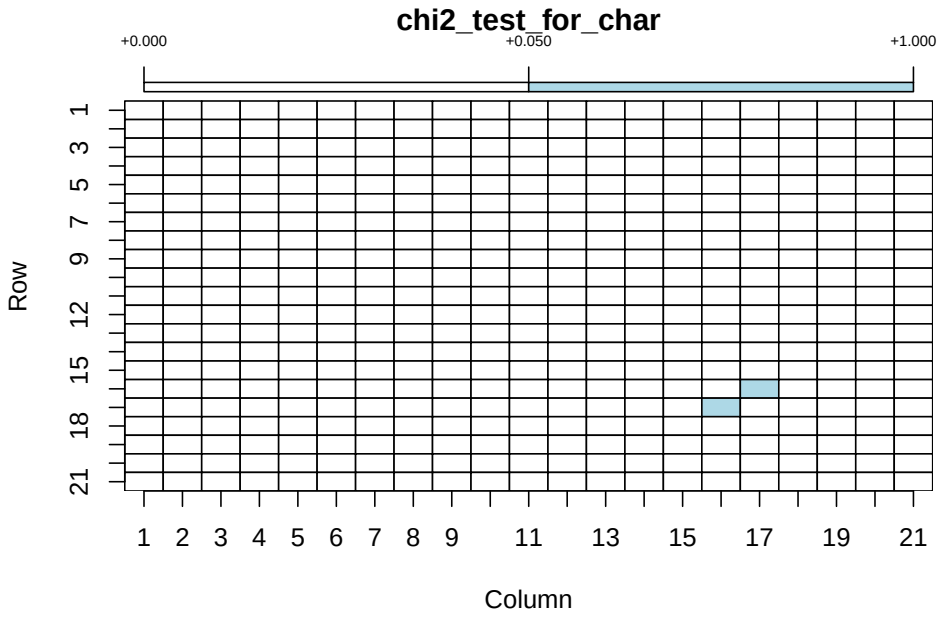
If $p.value > \alpha$: not significant result, fail to reject H_0, independent.

We'll set $p.value = 0.05$

3.3.2 Relationships between the characteristics by χ^2 test

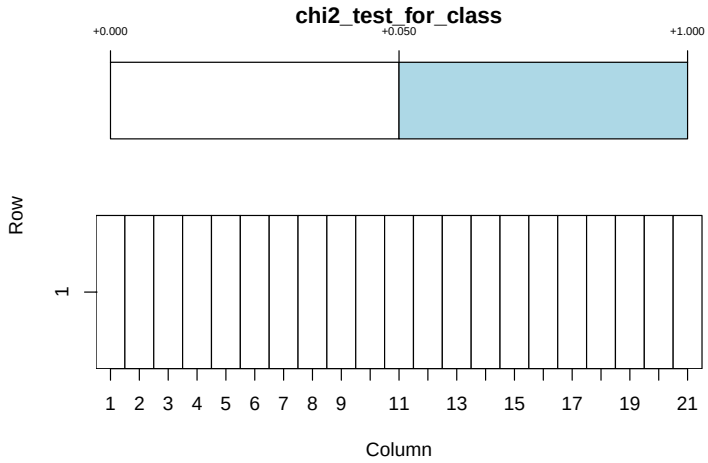
The rows and the columns represent the 21 features. The diagonal represents the correlation of the feature with itself (always correlated).

We can see that all the features correlated with each other except 'veil color' and 'ring number' that are independent from each other



3.3.3 Relationships between the characteristics to class by χ^2 test

All the characteristics are correlated with the mushroom class



4 Methods and Analysis

A classification algorithm is a function that weighs the input features so that the output separates one class into positive values and the other into negative values. Classifier training is performed to identify the weights that provide the most accurate and best separation of the two classes of data.

Among the common classification algorithms are classification/decision trees (ct), Random forest (rf), and K-nearest neighbor (KNN). The latter works slightly different than the first two.

Since the correlation matrix showed that the features are dependent, we will start with all the 21 features.

4.1 Model 1 - Classification trees

- Classification / decision trees, are used in prediction problems where the outcome is categorical. We form predictions by calculating which class is the most common among the training set observations within the partition.
- The default **splitting** method for classification called "Gini Impurity" ("Gini Index") and is the following:

$$Gini(j) = \sum_{k=1}^{K} \hat{p}_{j,k}(1 - \hat{p}_{j,k})$$

We define $\hat{p}_{j,k}$= the proportion of observation in partition j that are in class k

"Gini Impurity" seek to partition observation into subset that have the same class, when all observations in a group fall into single category, then $Gini(j) = 0$

- Train ct model and find best parameters

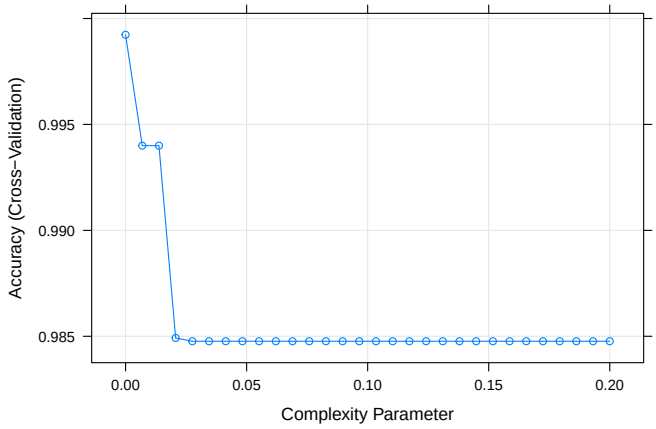
In order to avoid overfitting, we'll use 10-folds cross validation. The splits are done by 80%-20% training and test set.

```
train_control <- trainControl(method="cv", number = 10, p = 0.8) # use 10-folds cross-validation
tune_grid <- data.frame(cp = seq(0.0, 0.2, len=30)) # find the best value for cp

# Train using "caret"
train_ct <-
  train(train_set, y,
        method = "rpart",
        tuneLength = 6,
        trControl = train_control,
        tuneGrid = tune_grid)
# Find best cp value
train_ct$bestTune
```

```
##   cp
## 1  0
```

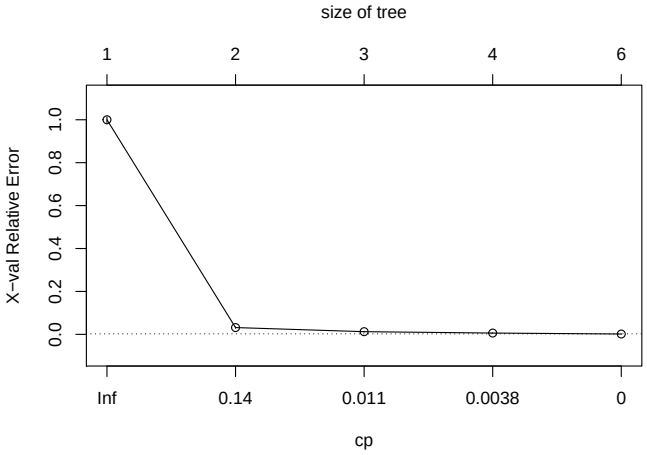
The complexity parameter (cp) is the minimum improvement in the model that is needed in each node (the Gini impurity must improve by a factor of cp for the new partition to be added).

When cp=0 then the tree is fully grown and no need to prune it. Large values of cp will force the algorithm to stop earlier which results in fewer nodes.



- Fit the model with the best complexity parameter

```
fit_ct <-
  rpart(y ~ .,
        data = train_set,
        method = "class",
        control = rpart.control(cp = train_ct$bestTune$cp , minsplit = 20))
```

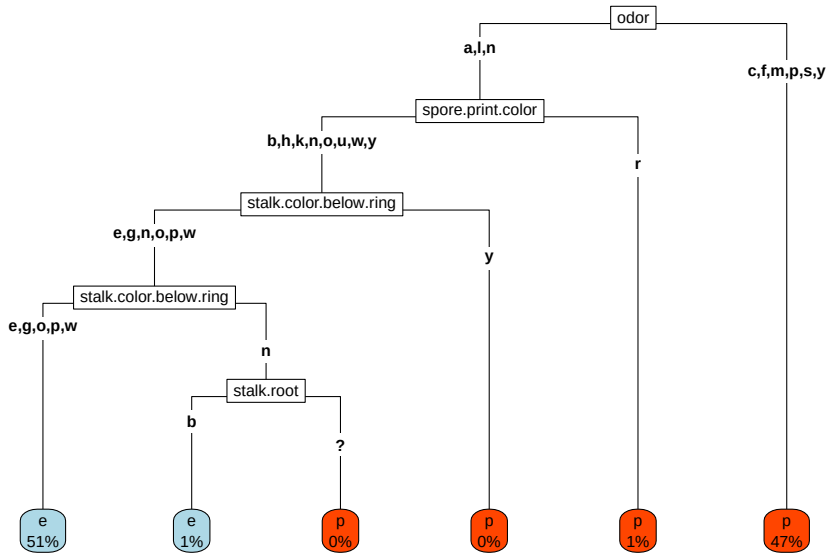


The above graph shows that the cross validation relative error (X-val) reaches the minimal value for a tree of size 6, which corresponds to a complexity parameter of 0.

The final tree is the following. The top node contain the entire sample, each of the remaining nodes contain a subset of the sample in the node directly above it.

As we predicted before, 'Odor' is a dominant feature and is the first to be splitted, the next feature to be splitted is 'Spore print color', while the last is 'Stalk root'.

Notice that the algorithm splitted 4 features out of 21.

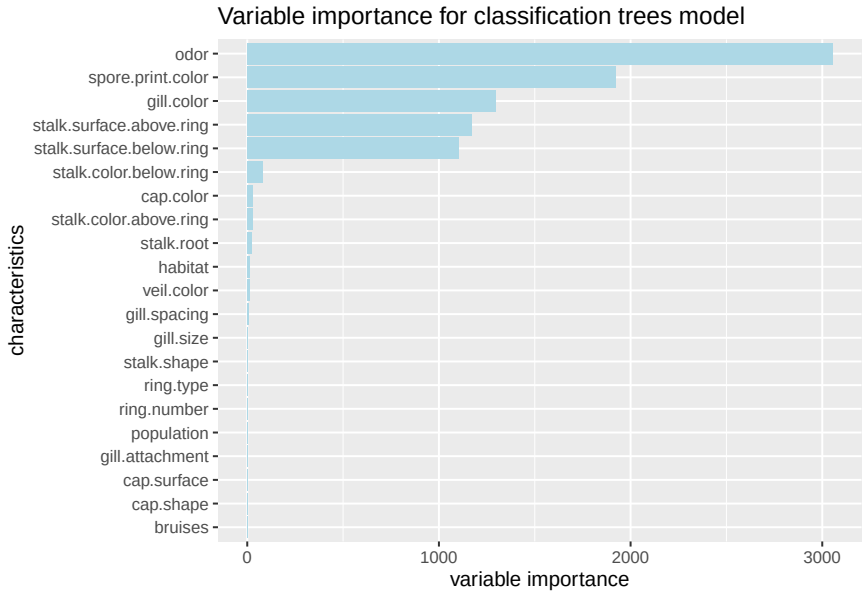


- Variable importance

The importance of the variable calculated by "Gini Importance" or "mean decrease gini"

"Gini importance" calculates each feature importance as the sum over the number of splits that include the feature, proportionally to the number of samples it splits.

We can interpret it as how valuable or needed this variable to the split. We can see that 8 out of the 21 features did not contribute any information to the splits , but "Stalk" characteristics had valuable influence overall.



- Predict the outcome

```
y_hat_ct <- predict(fit_ct, test_set, type = "class")
```

- Report accuracy

```
##            Reference
## Prediction   e   p
##          e 842   3
##          p   0 781
```

Accuracy	Sensitivity	Specificity	Prevalence	Balanced Accuracy
0.998155	1	0.9961735	0.5178352	0.9980867

- Model result

Classification trees model achieved "only" 0.998 of accuracy, with false-positive mistake of 3 edible observation that predicted as poisonous In true-life terms, this mistake is forgiven, but it wasn't if the opposite has occurred.

4.2 Model 1.1 - Classification trees - only "Visual Characteristics"

The previous model took into consideration all the variables and we saw that 2 of the general characteristics played a significant role in the splitting process and variable importance. Yet, we got accuracy of 99.8% and false-positive mistake of 3 observations.

Let's investigate a case which we are asking to classify a mushroom by an image. What would be the best features for the classification?

- Create new dataset, omit the general characteristics

```
visual_mushrooms <-
  mushrooms %>%
  select(-odor, -habitat, -population, -spore.print.color)
```

- We'll split the new data into training and test sets
- Train tuned ct model and find best parameters

We'll repeat the previous steps and train the model to find the best complexity parameter with 10-folds cross-validation to avoid overfitting.

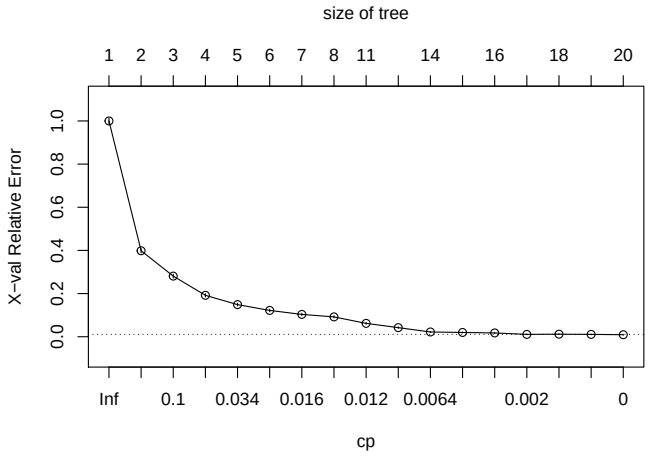
```
train_control <- trainControl(method="cv", number = 10, p = 0.8)
tune_grid <- data.frame(cp = seq(0.0, 0.2, len=30))
train_ct_visual <- train(train_visual, y_visual,
                  method = "rpart",
                  tuneLength = 6,
                  trControl = train_control,
                  tuneGrid = tune_grid)
# Find best cp value
train_ct_visual$bestTune
```

```
##   cp
## 1  0
```

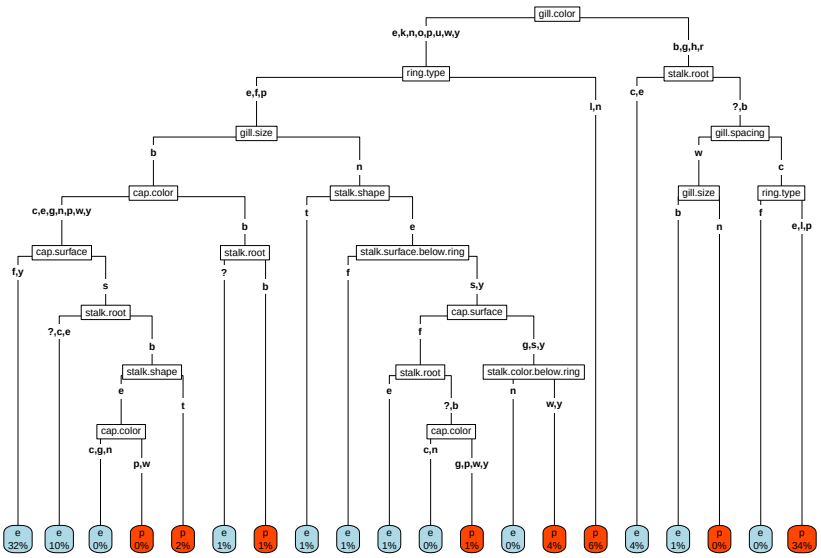
- Fit the model with best parameter

```
fit_ct_visual <-
  rpart(y_visual ~ .,
        data = train_visual,
        method = "class",
        control = rpart.control(cp = train_ct_visual$bestTune$cp , minsplit = 20))
```

The cross validation relative error (X-val) reaching the minimal value for a tree of size 20, which corresponds to a complexity parameter of 0. This tree pruned with 16 more branches than the original tree.

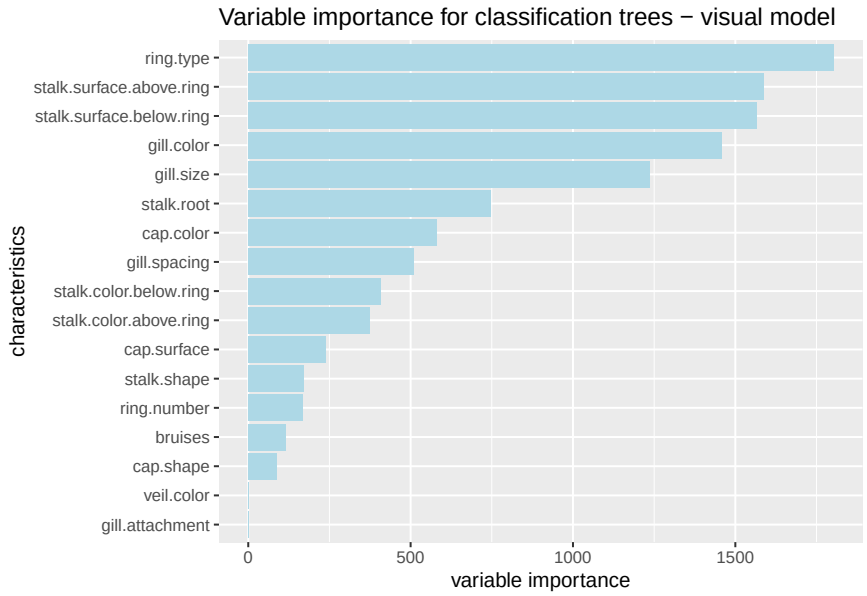


- The 'visual' tree



- Variable importance

Stalk still have large influence on the splitting, while ring type has the highest 'mean decrease Gini' (used to be zero in the original tree), means that gill color (the first var. to split) and the ring type is highly correlated.



- Predict the outcome

```
y_hat_ct_visual <- predict(fit_ct_visual, test_visual, type = "class")
```

- Report accuracy

```
##            Reference
## Prediction   e   p
##          e 841   5
##          p   1 779
```

Accuracy	Sensitivity	Specificity	Prevalence	Balanced Accuracy
0.99631	0.9988124	0.9936224	0.5178352	0.9962174

- Model results

The visual model achieved lower result than the original - 99.6% accuracy.

There is 5 false-positive results (5 edible predicted poisonous) and one false-negative result (poisonous predicted as edible).

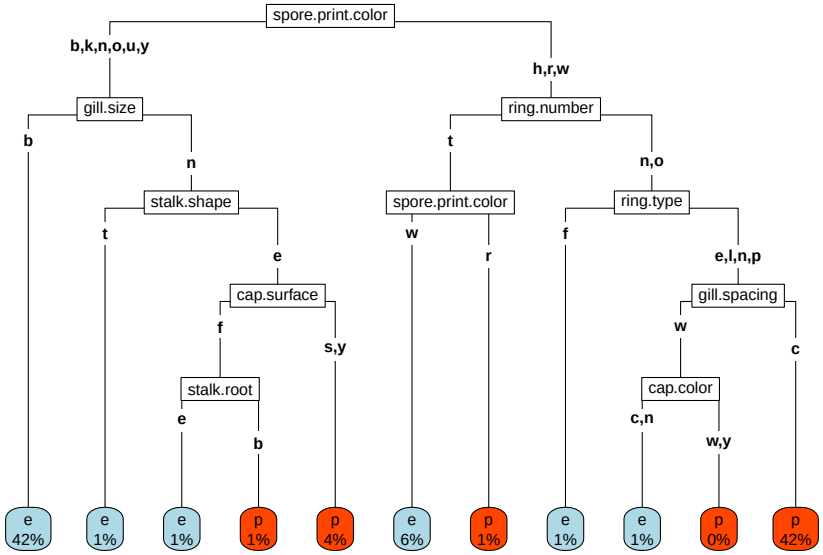
Means, one of the general characteristics is essential for the classification process.

4.3 Model 1.2 - Classification trees - tuned

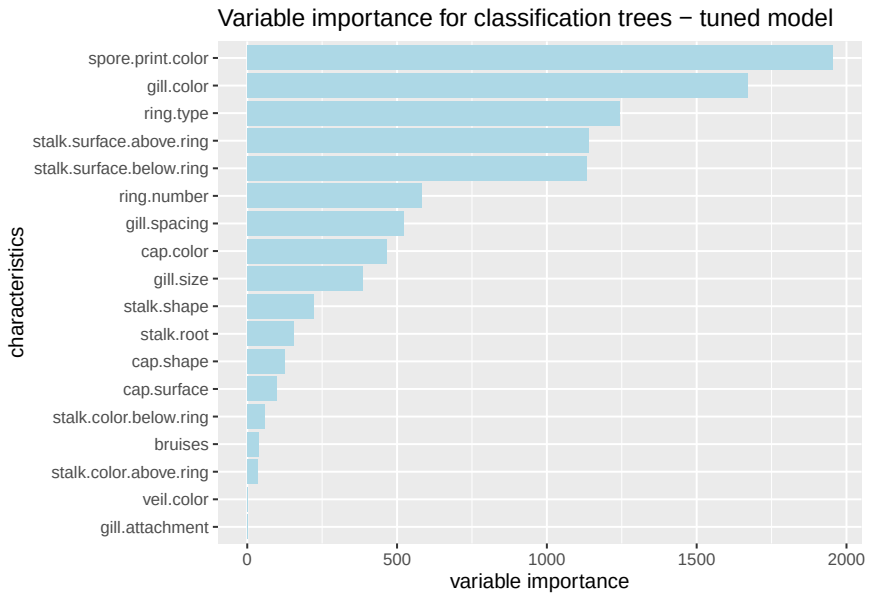
- Create new dataset, omit the general characters, return spore.print.color

```
tuned_mushrooms <-
  mushrooms %>%
  select(-odor, -habitat, -population)
```

- Train tuned ct model and find best parameters
- Plot the tree



- Graph variable importance



- Predict and report accuracy

```
##          Reference
## Prediction   e   p
##          e 842   0
##          p   0 784
```

Accuracy	Sensitivity	Specificity	Prevalence	Balanced Accuracy
1	1	1	0.5178352	1

- Model results

The model achieved a perfect accuracy of 100%. It can be assumed that 'Odor' that was a dominant feature in the original model was "stand in the way" towards a perfect prediction. Smell can be very confusing sometimes.

In addition, the model suggests that 'gill attachment', 'veil color', and 'veil type' do not contribute information to the model and therefore are not needed, alongside with 'odor', 'population' and 'habitat'.

- Advantages of Classification trees

Classification trees are highly interpretable and easy to visualize (if small enough). They can model human decision processes and don't require use of dummy predictors for categorical variables.

- Classification trees limitations

The approach via recursive partitioning can easily over-train and is therefore a bit harder to train. In terms of accuracy, it is rarely the best performing method since it is not very flexible and is highly unstable to changes in training data. That is why we saw such big differences in the above models (particularly in the size of the trees), when we omitted/added features.

4.4 Model 2 - Random Forest

The Random forests combines multiple Classification trees. Its goal is to improve prediction performance and reduce instability by averaging multiple Classification trees.

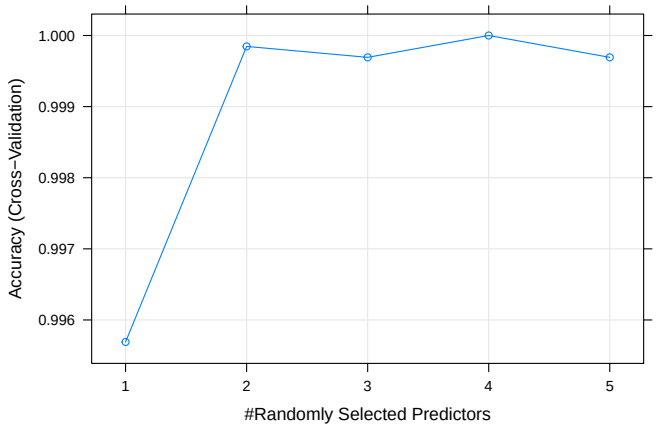
- Train rf model and find best parameters

Same here, we will use 10-folds cross validation and set a tune_grid for "mtry" (number of variables randomly sampled as candidates at each split). We can see in the next plot that the accuracy is higher when "mtry" reaches to 4 randomly sampled variables.

```
set.seed(3, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(3)`

train_control <- trainControl(method="cv", number = 10, p = 0.8) # use 10-folds cross-validation
tune_grid <- data.frame(mtry =  c(1:5)) # find the best value for tuning mtry

set.seed(13, sample.kind="Rounding") # if using R 3.5 or earlier, use 'set.seed(13)'
```

```
train_rf <-
  train(train_set, y,
        method = "rf",
        ntree = 200,  #number of trees to grow
        trControl = train_control,
        tuneGrid = tune_grid)
# Find best mtry parameter
train_rf$bestTune
```

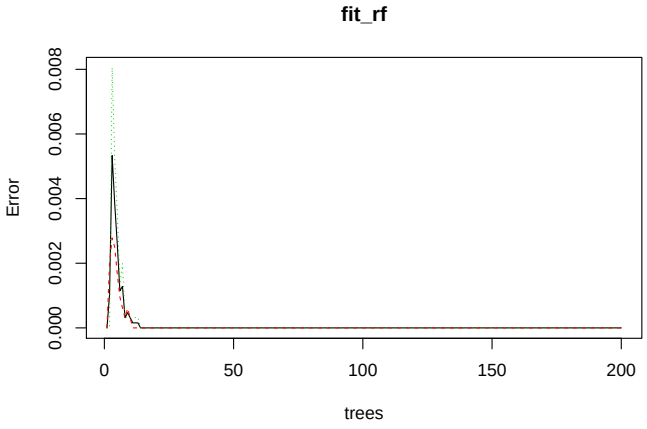
```
##   mtry
## 4    4
```



- Fit the model with best parameter

```
fit_rf <-
  randomForest(train_set, y,
               ntree = 200,
               nodesize = train_rf$bestTune$mtry)
```

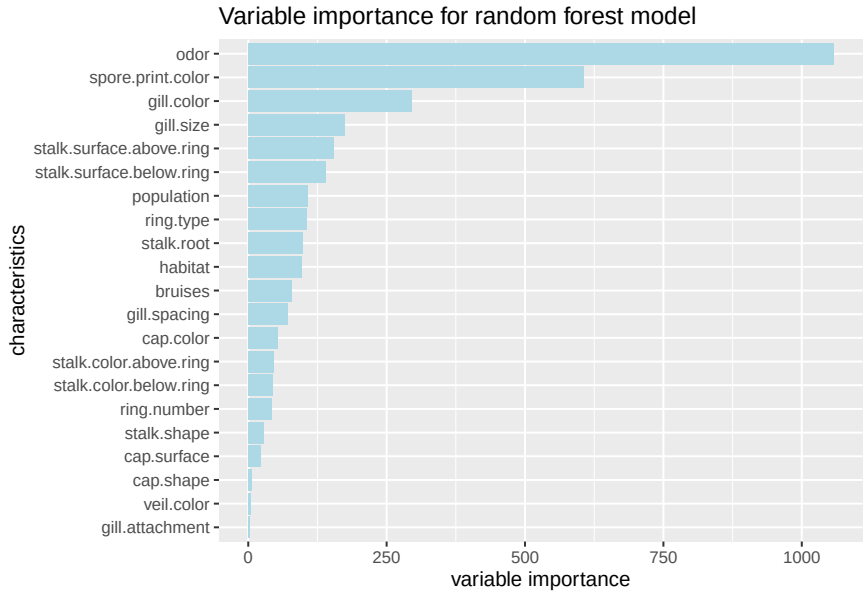
The accuracy improves as we added more trees until about 20 trees where accuracy stabilized



Random forest calculates the variable importance over ensembles of randomized trees.

The Random Forest first important variable is 'Odor'. In addition, all general characteristics are participating in the classification and have a predictive power.

Down the list, the less important features 'gill attachment' and 'veil color'. Those 2 features also weren't needed in the tuned ct model that got 100% accuracy. Meaning, we can omit them.



- Predict the outcome

```
y_hat_rf <- predict(fit_rf, test_set, type = "class")
```

- Report accuracy

```
##            Reference
## Prediction   e   p
##          e 842   0
##          p   0 784
```

Accuracy	Sensitivity	Specificity	Prevalence	Balanced Accuracy
1	1	1	0.5178352	1

- Model results

Random forest achieved 100% accuracy.

Random forest chooses features randomly during the training process. Therefore, it does not depend highly on any specific set of features. It is more difficult to interpret than a single classification tree, but the randomized feature selection makes random forest much more accurate than a decision tree.

Therefore, rf could handle 'Odor' better than the simple first model of classification trees.

In addition, both rf and ct did not need 'gill attachment' and 'veil color' as a predictors for a perfect accuracy.

- Random forest limitations

The more accurate prediction requires more trees while large number of trees can make the algorithm too slow and ineffective for real-time predictions.

With 10-folds cross validation, the overall runtime of the model was 24 seconds: 21 seconds for training, less than one second for fitting, and less than one second for predicting.

When dealing with big amount of data to train and predict, it's faster to use the 'Rborist' package rather than 'randomForest'

4.5 Model 3 - k-nearest neighbors

K-nearest neighbors is a pattern recognition algorithm that uses training datasets to find the k closest relatives in future examples.

When KNN is used in classification with categorical data, the basic idea is that each category is mapped into a real number (dummy) in some optimal way, and then KNN classification is performed using those numeric values.

- For this model will use the conclusions from the previous models that achieved 100% accuracy and omit the two unnecessary features.
- Creating a new data set

Since KNN cannot handle characters, we'll convert the mushrooms data set into canonical data set, as the following overview. Then, we'll split the data into training and test sets.

```
##    class cap.shape cap.shape cap.shape cap.shape cap.shape cap.shape
## 1      p         0         0         0         0         0         1
## 2      e         0         0         0         0         0         1
## 3      e         1         0         0         0         0         0
## 4      p         0         0         0         0         0         1
## 5      e         0         0         0         0         0         1
## 6      e         0         0         0         0         0         1
## 7      e         1         0         0         0         0         0
## 8      e         1         0         0         0         0         0
## 9      p         0         0         0         0         0         1
## 10     e         1         0         0         0         0         0
```

- Train KNN model and find best parameters

We'll use here 5-folds cross-validation to avoid overfitting, since setting the number of folds to 10 increased the model runtime in 140% and achieved the same model accuracy.

In addition, we'll set a tune_grid for K in order to find the optimal parameter

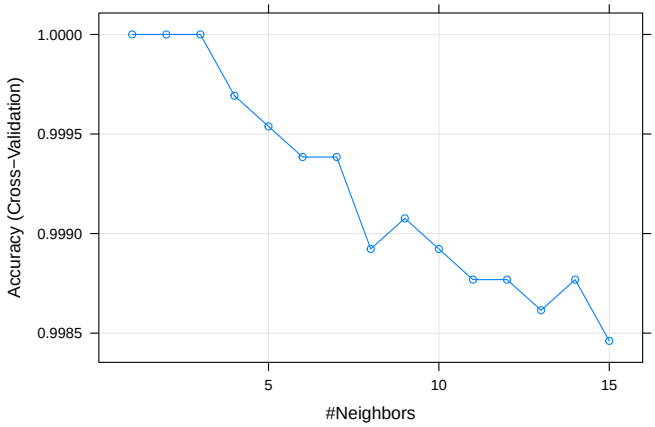
```
set.seed(201, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(201)`
train_control <- trainControl(method="cv", number = 5, p = 0.8)
tune_grid <- data.frame(k = seq(1, 15, by=1))

set.seed(74, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(74)`

train_knn <-
```

```
  train(train_set, y,
        method = "knn",
        trControl = train_control,
        tuneGrid = tune_grid)
# find optimal k
train_knn$bestTune
```

```
##   k
## 3 3
```

The model achieved the highest accuracy with k=3, meaning 3 variables in each neighborhood.

In case of low accuracy, we can suspect the lowest K as overtraining of the model



- Fit the model with best parameter

```
fit_knn <- knn3(train_set, y,
                k = train_knn$bestTune$k)
```

- Predict the outcome and report accuracy

```
##            Reference
## Prediction   e   p
##          e 842   0
##          p   0 784
```

Accuracy	Sensitivity	Specificity	Prevalence	Balanced Accuracy
1	1	1	0.5178352	1

- Model results

The low K=3 could lead us to an overtraining of the model, but even though and with the omission of two unnecessary features, KNN achieved 100% accuracy.

- limitation

1. With 5-folds cross validation, the overall runtime of the model was 97 seconds: 94 seconds for training, less than one second for fitting, and two seconds for predicting (10-folds cv increased the total runtime in 40 seconds and achieved the same model accuracy)
2. Since KNN is based on the Euclidean distances, it unable to be applied directly on categorical data. In order to define the distance metrics for categorical variables, we need to preprocess the dataset by transforming it to dummy variables that represent the categorical variables.

5 The Models Results

5.1 The models accuracy

Random forest, KNN and Classification trees (1.2 tuned model - variables omitted) achieved the perfect 100% accuracy.

The first two Classification trees models (1, 1.1) achieved high, but not enough, accuracy while the latter got a fatal false-negative mistake by classifying poisonous as edible.

	Accuracy
Classification Trees	0.998155
Classification Trees - only 'Visual Characteristics'	0.996310
Classification Trees - tuned	1.000000
Random Forest	1.000000
KNN	1.000000

5.2 The 15 most important features according to the 1.2 Classification trees model and Random forest

The next table represent the top 15 important variables (out of initial 21).

Each one of the following models consider different features as most important.

Yet, 'spore.print.color' and 'gill.color' have large contribution to both of the models (those are highly correlated, since the gill is the part that produces the spore and it is only reasonable to share the color).

In addition, 'gill.attachment' and 'veil.color' omitted from KNN and Classification trees model while playing a very negligible, if any, role in random forest classifier.

```
##        Classification.trees            Random.forest
## 1         spore.print.color                     odor
## 2                gill.color        spore.print.color
## 3                 ring.type               gill.color
## 4  stalk.surface.above.ring                gill.size
## 5  stalk.surface.below.ring stalk.surface.above.ring
## 6               ring.number stalk.surface.below.ring
## 7              gill.spacing               population
## 8                 cap.color                ring.type
## 9                 gill.size               stalk.root
## 10              stalk.shape                  habitat
## 11               stalk.root                  bruises
```

```
## 12                cap.shape             gill.spacing
## 13              cap.surface                cap.color
## 14   stalk.color.below.ring   stalk.color.above.ring
## 15                  bruises   stalk.color.below.ring
```

6 Conclusion

- What types of machine learning models perform best on this dataset?

Analyzing and modeling the Mushrooms classification dataset lead us to a conclusion that models such as KNN and Random forest, which are more flexible and easier to train and tune, are better for classifying.

The limitation of each of the KNN and rf models that discussed at the end of each section, did not hurt or biased the results. In addition, rf performed better runtime- 422% faster than KNN runtime.

For classification trees, in my opinion, the model couldn't handle well with the task, since we needed some rough adjustments, including omitting the most important var ('Odor') in order to achieve 100% accuracy. Second, the model was very sensitive to changes and changed the size of the trees in a significant way.

- Which features are most indicative of a poisonous mushroom?

We can see that omitting 'gill.attachment' and 'veil.color' from the models led us to 100% accuracy.

The common 4 top important features from Classification trees (1.2) model and rf model are 'spore.print.color', 'gill.color', 'stalk.surface.above.ring', 'stalk.surface.below.ring' and we must not omit them.

- Future work

As for the 'Odor', there is no certainty about its roll.

I'd like to test some more classification algorithms such as 'Logistic Regression' and 'Support Vector Machines' to better determine which algorithms perform best under their limitations and create a better stable list of features importance.