Project

Talha Mahin Mir

May 15, 2017

Data Cleaning

```
mushrooms = read.csv("C:/Users/Talha Mir/Desktop/Course Slides/Semester 2/Data Mining/Project/mushrooms
nrow(mushrooms)
```

```
## [1] 8124
```

```
ncol(mushrooms)
```

```
## [1] 23
```

```
str(mushrooms)
```

```
## 'data.frame':    8124 obs. of  23 variables:
##  $ class                   : Factor w/ 2 levels "e","p": 2 1 1 2 1 1 1 1 2 1 ...
##  $ cap.shape               : Factor w/ 6 levels "b","c","f","k",..: 6 6 1 6 6 6 1 1 6 1 ...
##  $ cap.surface             : Factor w/ 4 levels "f","g","s","y": 3 3 3 4 3 4 3 4 4 3 ...
##  $ cap.color               : Factor w/ 10 levels "b","c","e","g",..: 5 10 9 9 4 10 9 9 9 10 ...
##  $ bruises                 : Factor w/ 2 levels "f","t": 2 2 2 2 1 2 2 2 2 2 ...
##  $ odor                    : Factor w/ 9 levels "a","c","f","l",..: 7 1 4 7 6 1 1 4 7 1 ...
##  $ gill.attachment         : Factor w/ 2 levels "a","f": 2 2 2 2 2 2 2 2 2 2 ...
##  $ gill.spacing            : Factor w/ 2 levels "c","w": 1 1 1 1 2 1 1 1 1 1 ...
##  $ gill.size               : Factor w/ 2 levels "b","n": 2 1 1 2 1 1 1 1 2 1 ...
##  $ gill.color              : Factor w/ 12 levels "b","e","g","h",..: 5 5 6 6 5 6 3 6 8 3 ...
##  $ stalk.shape             : Factor w/ 2 levels "e","t": 1 1 1 1 2 1 1 1 1 1 ...
##  $ stalk.root              : Factor w/ 5 levels "?","b","c","e",..: 4 3 3 4 4 3 3 3 4 3 ...
##  $ stalk.surface.above.ring: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
##  $ stalk.surface.below.ring: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
##  $ stalk.color.above.ring  : Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...
##  $ stalk.color.below.ring  : Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...
##  $ veil.type               : Factor w/ 1 level "p": 1 1 1 1 1 1 1 1 1 1 ...
##  $ veil.color              : Factor w/ 4 levels "n","o","w","y": 3 3 3 3 3 3 3 3 3 3 ...
##  $ ring.number             : Factor w/ 3 levels "n","o","t": 2 2 2 2 2 2 2 2 2 2 ...
##  $ ring.type               : Factor w/ 5 levels "e","f","l","n",..: 5 5 5 5 1 5 5 5 5 5 ...
##  $ spore.print.color       : Factor w/ 9 levels "b","h","k","n",..: 3 4 4 3 4 3 3 4 3 3 ...
##  $ population              : Factor w/ 6 levels "a","c","n","s",..: 4 3 3 4 1 3 3 4 5 4 ...
##  $ habitat                 : Factor w/ 7 levels "d","g","l","m",..: 6 2 4 6 2 2 4 4 2 4 ...
```

```
sum(is.na(mushrooms))
```

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

Origina data has x rows and y features and no NA values. Analyzing the indivvidual features we can see that veil.type has only 1 level, so it's not essential in analyzing the problem. Dropped it.

```
mushrooms$veil.type<-NULL
```

Also stalk.root has some missing values "?". Removing rows with missing values as well, as in this case imputing missing values can be fatal.

```
mushrooms <- mushrooms[-which(mushrooms$stalk.root == "?"), ]
```

Final data set has x rows and y columns.

```
nrow(mushrooms)
```

```
## [1] 5644
```

```
ncol(mushrooms)
```

```
## [1] 22
```

Application of various algorithms and identification of important features

Data Splitting

Typical 80-20 split

```
sample <- sample.int(nrow(mushrooms), floor(.80*nrow(mushrooms)), replace = F)
mushroomsDatatrain <- mushrooms[sample, ]
mushroomsDatatest <- mushrooms[-sample, ]
```

ID3

As ID3 starts with identifying the features that has high information gain, so we thought this would help identify the most prominent features for classiying the mushrooms.

```
library(RWeka)
```

```
## Warning: package 'RWeka' was built under R version 3.3.3
```

```
library(partykit)
```

```
## Warning: package 'partykit' was built under R version 3.3.3
## Loading required package: grid
```

```
fit <- J48(class~., data=mushroomsDatatrain)
predictions <- predict(fit, mushroomsDatatest)

testDataValues = as.numeric(mushroomsDatatest$class)-1
predictedValues = as.numeric(predictions)-1

id3Precision <- sum(predictedValues & testDataValues) / sum(predictedValues)
id3Recall <- sum(predictedValues & testDataValues) / sum(testDataValues)

id3Precision
```

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

```
id3Recall
```

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

Precision and recall values are 1. Awesome!

visualization of the obtained tree:

```r
#plot(as.party(fit))
# Don't know why this stupid function is not working in RMarkdown, including pic below
```

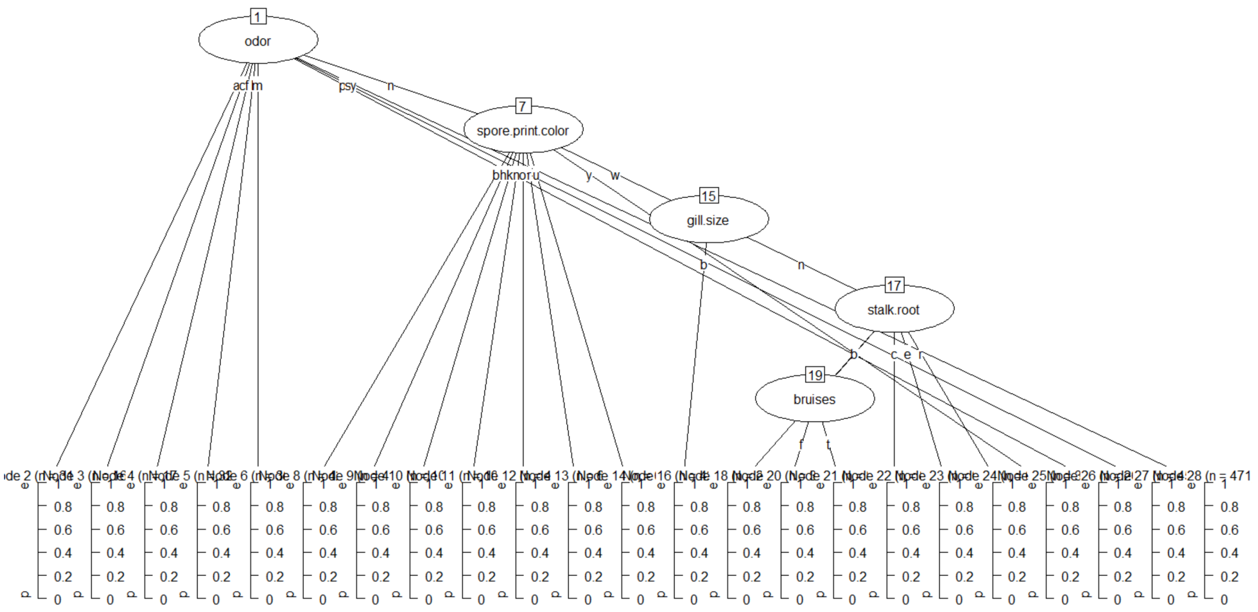


Figure 1:

We can see that odor and spore color are most important features.

To further confirm it, we printed out information gain for all the features:

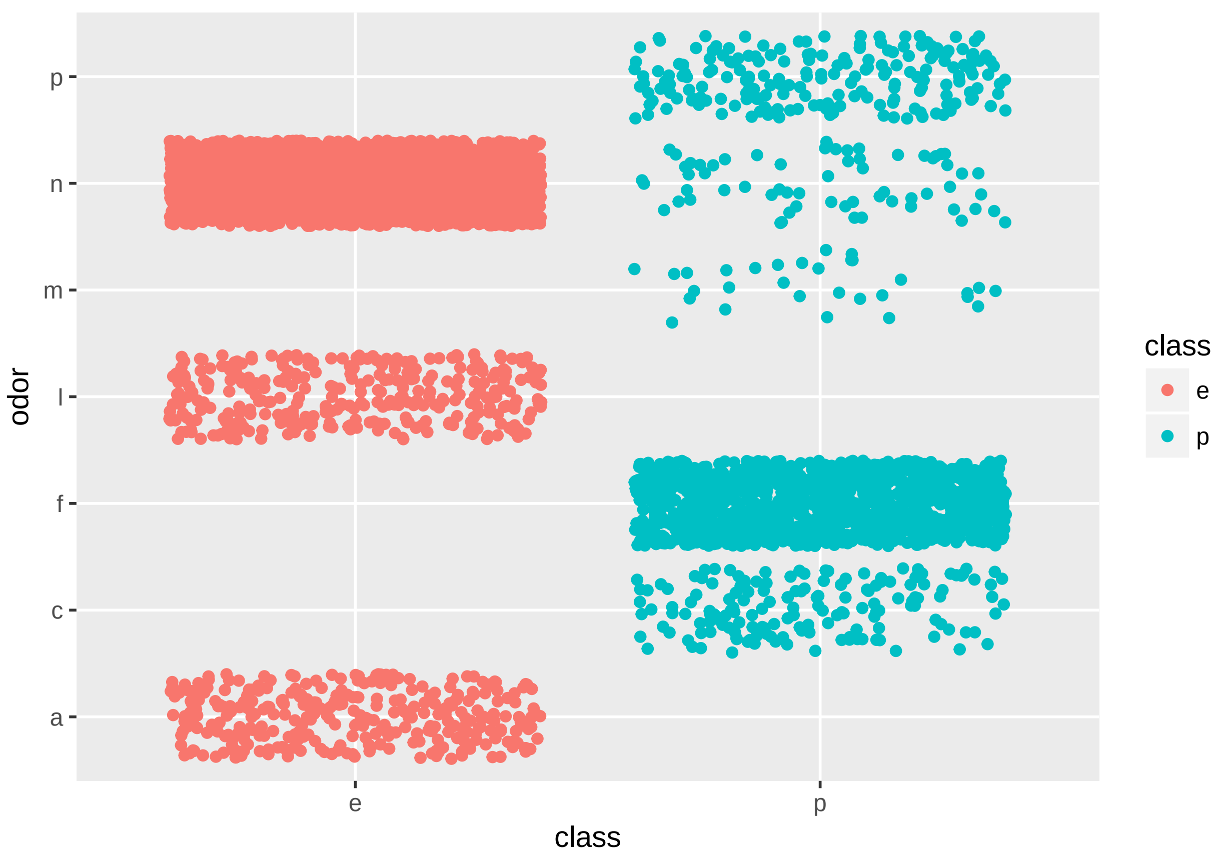
```r
InfoGainAttributeEval(class ~ . , data = mushroomsDatatrain)
```

```
##                 cap.shape              cap.surface                cap.color
##               0.016056870              0.003295582              0.190934155
##                   bruises                     odor          gill.attachment
##               0.139161465              0.865431095              0.004649838
##              gill.spacing                gill.size               gill.color
##               0.061003728              0.032028611              0.212964834
##               stalk.shape               stalk.root stalk.surface.above.ring
##               0.271564371              0.101102695              0.422997031
## stalk.surface.below.ring   stalk.color.above.ring   stalk.color.below.ring
##               0.406864610              0.306149744              0.276524464
##                veil.color              ring.number                ring.type
##               0.001547044              0.012963961              0.464554089
##         spore.print.color               population                  habitat
##               0.582138928              0.113256678              0.099711570
```

Odor has the highest score: 0.906074977

Illustration of the relationships between ‘class’ and ‘odor’:

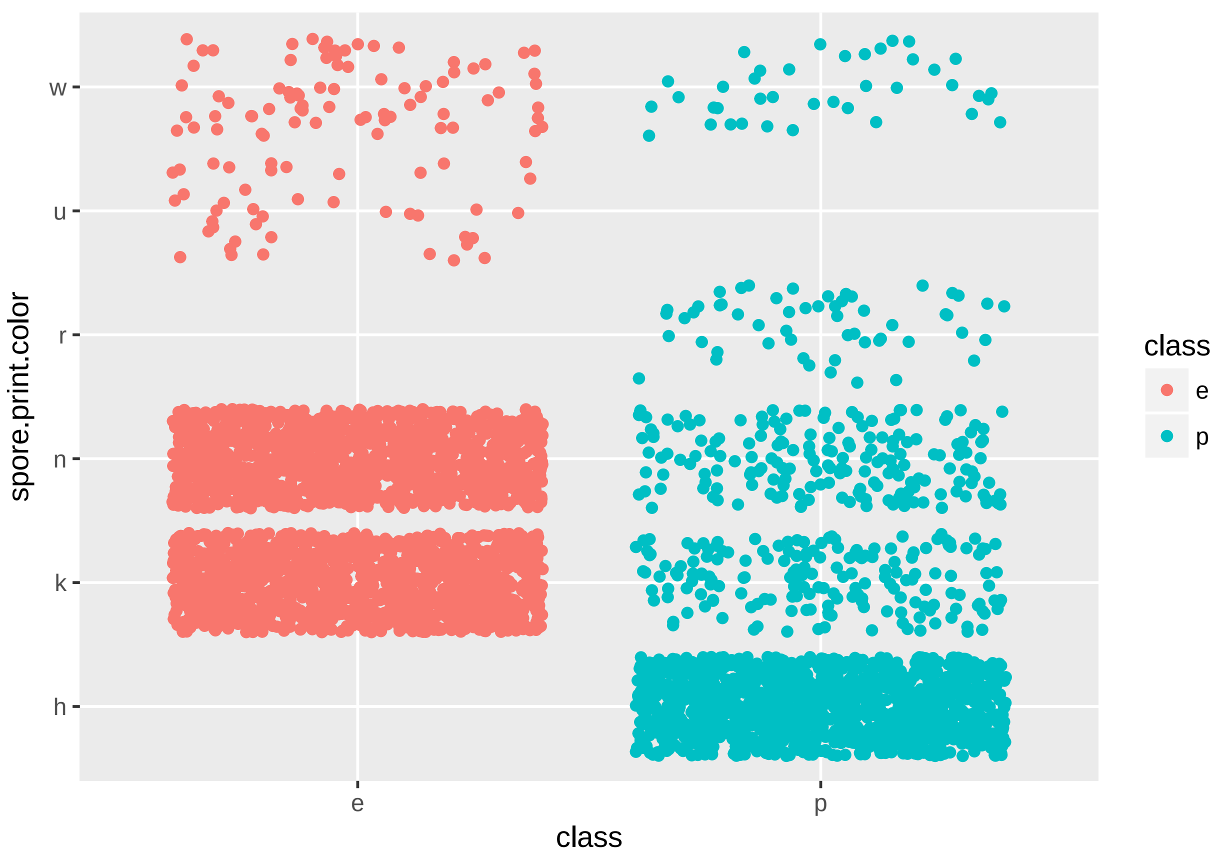
```r
library(ggplot2)
p <- ggplot(mushroomsDatatrain, aes(x=class,y=odor,color=class), alpha=0.3) + geom_jitter()
p
```



Description on it later.

The second most valuable feature is spore.print.color. Illustration:

```
p <- ggplot(mushroomsDatatrain, aes(x=class,y=spore.print.color, color=class), alpha=0.3) + geom_jitter
p
```



Description later.

To compare the performance and further confirm our results we used a couple of other techniques we learn during the course.

Random Forest

```r
library("ROCR")
```

```
## Warning: package 'ROCR' was built under R version 3.3.3

## Loading required package: gplots

## Warning: package 'gplots' was built under R version 3.3.3

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.3.3

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
mushrooms_randomForest <- randomForest(class~., data=mushroomsDatatrain)

predictions <- predict(mushrooms_randomForest, mushroomsDatatest)

testDataValues = as.numeric(mushroomsDatatest$class)-1
predictedValues = as.numeric(predictions)-1

rFPrecision <- sum(predictedValues & testDataValues) / sum(predictedValues)
rFRecall <- sum(predictedValues & testDataValues) / sum(testDataValues)

rFPrecision
```

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

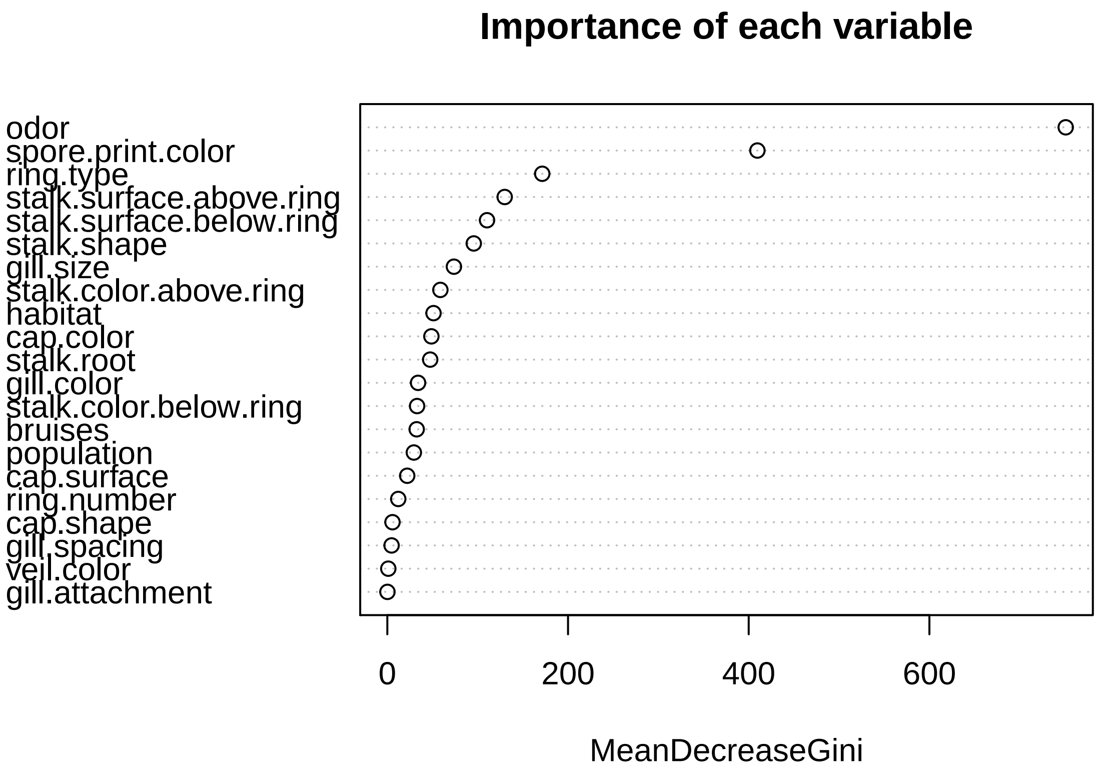
```
rFRecall
```

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

Precision and recall values are 1 again.

Variable importance chart from RF:

```
dataimp <- varImpPlot(mushrooms_randomForest, main = "Importance of each variable")
```



Description later.

Naive Bayes

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.3.3
```

```r
mushroom_naive_bayes = naiveBayes(class~., data=mushroomsDatatrain)
predictions <- predict(mushroom_naive_bayes, mushroomsDatatest)

predictions <- predict(mushroom_naive_bayes, mushroomsDatatest)

testDataValues = as.numeric(mushroomsDatatest$class)-1
predictedValues = as.numeric(predictions)-1

nBPrecision <- sum(predictedValues & testDataValues) / sum(predictedValues)
nBRecall <- sum(predictedValues & testDataValues) / sum(testDataValues)

nBPrecision
```

```
## [1] 0.9868074
```

```r
nBRecall
```

```
## [1] 0.85
```

In this case precision and recall are a bit less than 100%.