Coursework (CM3111)
Big Data Analytics

Shaw Eastwood, s.eastwood@rgu.ac.uk

Monday 18th December, 2017

Introduction
In this report I will detail my choice of dataset and the process of creating a model that fits the dataset.

1 Data Exploration

1.1 Dataset Choice

For the dataset I opted to use the Mushroom Classification dataset provided by UCI Machine Learning on Kaggle[1]. I chose this dataset because of its potential practical application in predicting the edibility of a mushroom merely based on characteristics without needing to know the name, merely by describing it can it be identified. This dataset was also featured on Kaggle and came very well recommended. The size of the dataset also made it a suitable choice it was a manageable size with a high variance in the attributes.

1.2 Technology Platform

The use of big data exploration tools such as Hadoop seemed to be of little merit as the size of this dataset does not warrant it. The source csv is a mere 374 kB with a very manageable 8124 rows of data. In addition the variables are single letters representing certain aspects, this helps cut down on size even further and means we don't need to run any matching or regex on the fields to find our data. All of this lead me to discard any such big data technologies from consideration.

1.3 Problem Statement & Data Exploration

1.3.1 Description

The dataset contains a list mushrooms and describes their various characteristic, listed below, along with whether or not they are poisonous or not. The aim of this report is to build a model which will predict to degree of certainty which characteristics of the mushrooms dictate the edibility of the mushrooms.

1.3.2 Attribute Information

classes: edible=e, poisonous=p

cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s

cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s

[1] https://www.kaggle.com/uciml/mushroom-classification/

cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y

bruises: bruises=t,no=f

odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s

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=

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

veil-type: partial=p,universal=u

veil-color: brown=n,orange=o,white=w,yellow=y

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

spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y

population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y

habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

[2]

[2]taken from the data page found here https://www.kaggle.com/uciml/mushroom-classification/data

Loading in the dataset:

```
options(scipen = 999)
setwd('~/Documents/rstuff/quitelargedata')
df <- read.csv('mushrooms.csv',header = T)
```

1.3.3 Number of Rows and Columns

```
dim(df)
```

```
## [1] 8124   23
```

```
cat('Number of Rows in the set is: ', nrow(df))
```

```
## Number of Rows in the set is:  8124
```

```
cat('Number of Columns/Features in the set is: ', ncol(df))
```

```
## Number of Columns/Features in the set is:  23
```

1.3.4 Names of Features

```
names(df) # names of the columns/features
```

```
##  [1] "class"                     "cap.shape"
##  [3] "cap.surface"               "cap.color"
##  [5] "bruises"                   "odor"
##  [7] "gill.attachment"           "gill.spacing"
##  [9] "gill.size"                 "gill.color"
## [11] "stalk.shape"               "stalk.root"
## [13] "stalk.surface.above.ring" "stalk.surface.below.ring"
## [15] "stalk.color.above.ring"   "stalk.color.below.ring"
## [17] "veil.type"                "veil.color"
## [19] "ring.number"              "ring.type"
## [21] "spore.print.color"        "population"
## [23] "habitat"
```

1.3.5 Class / Label Distribution

```
table(df$class) # show the distribution
```

```
##
##    e    p
## 4208 3916
```

1.3.6 Glance at the Data

```
dfs <- df[1:4,] # copy the first four rows into a temporary
names(dfs) <- NULL # delete the names of columns so the text output isn't massive
head(dfs) # show what the data looks like
```

```
##
## 1 p x s n t p f c n k e e s s w w p w o p k s u
## 2 e x s y t a f c b k e c s s w w p w o p n n g
## 3 e b s w t l f c b n e c s s w w p w o p n n m
```

```
## 4 p x y w t p f c n n e e s s w w p w o p k s u
```

1.3.7 Distribution

```
# plot the amount of edible vs. poisonous mushrooms in the set
barplot(
        table(df$class), col = c('lightblue', 'grey'),
        names.arg = c('edible', 'poisonous'), xpd = FALSE,
        ylim = c(3500, 4500), offset(3000), main = "Edible vs. Poisonous")
```

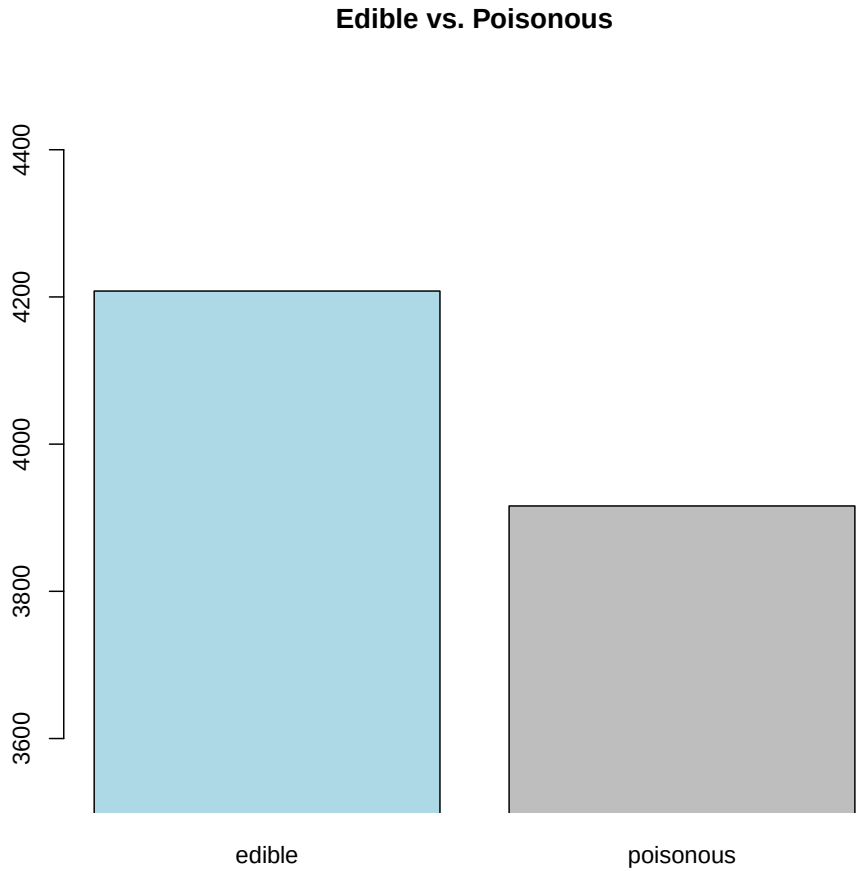

Figure 1: Amount of edible vs. poisonous in the data set

1.3.8 Graphing

```
library(caret)
# define the layout of the graphs
xy <- list(x=list(relation='free'), y=list(relation='free'))
```

```r
# plot the data on a density graph
# this works by converting all of the features except the class to a number
#  then using the class as the class
featurePlot(
        sapply(df[,-c(df$class)], function (x) as.numeric(x)),
        df$class, plot='density', scales=xy, layout = c(4,6),
        pch = '|', frame = F)
```

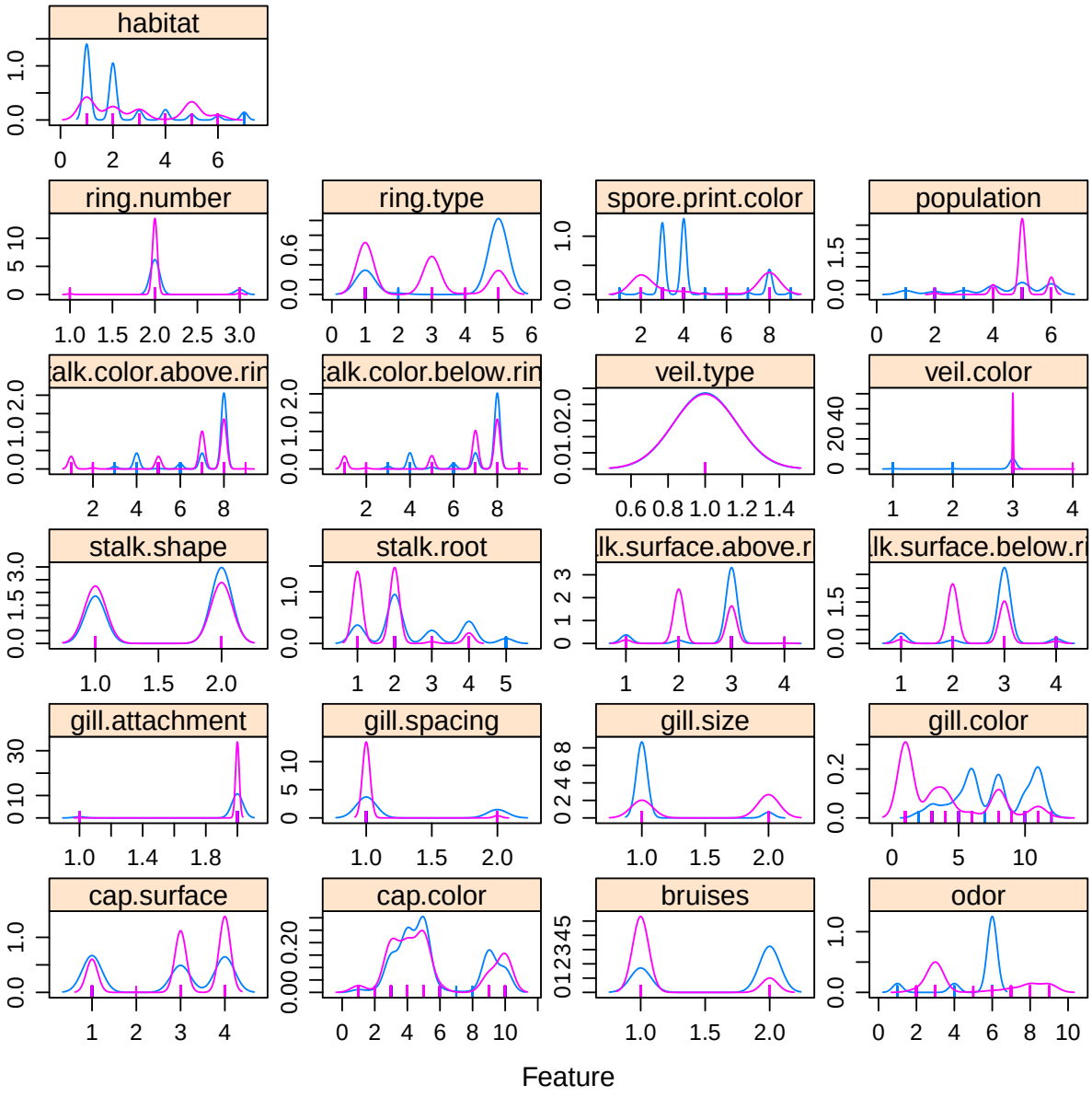

Figure 2: Correlation between edible and poisonous by feature

As the graphs above show there is no perfect separation between the features. The best place to find any separation are the features odor, spore.print.color, ring.type, population, habitat. A number of the values show a very close relationship and this are of little consequence in judging the class such as, like veil.type, cap.shape, cap.color and gill.attachment.

```
ggplot(data = df, aes(x = spore.print.color, y = odor, color = class)) +
        geom_jitter() + scale_color_manual(
                values = c('lightblue', 'grey')
        )
```

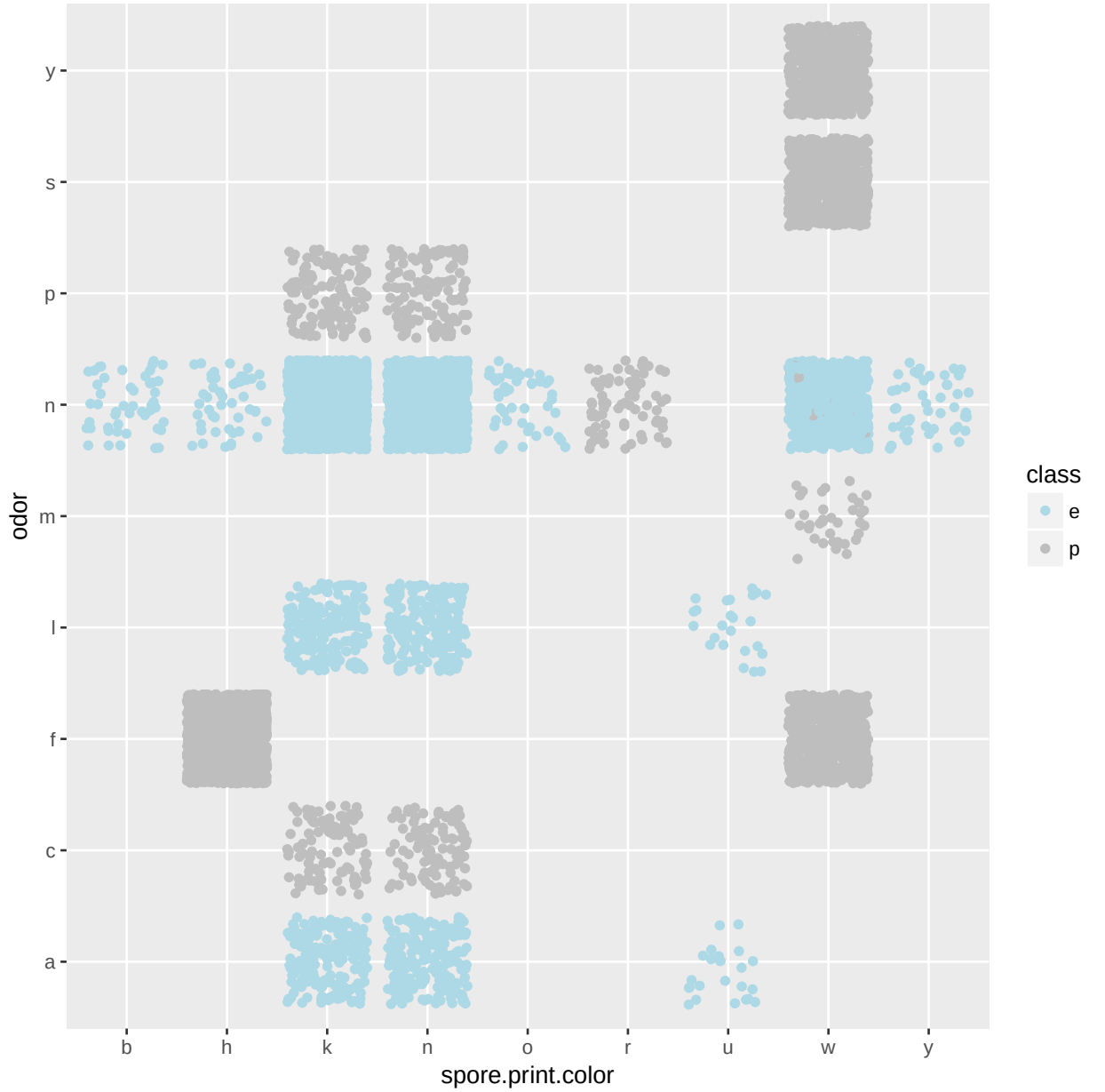

Figure 3: Showing the disparity between odor and spore.print.color

If we plot our two most prevalent features we can see they have very clear separation and will be the primary features our model uses to determine the class.

1.4 Pre-proccessing

```
length(unique(is.na(df))) # the length of the unique values for each col
```

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

Since our dataset doesn't have any missing values we don't need to worry about handling and null values or missing rows.

```
# loop through all of the features and show the amount of uniqe values
for (i in names(df)) { cat (i, ':', length(unique(df[[i]])), '\n')}
```

```
## class : 2
## cap.shape : 6
## cap.surface : 4
## cap.color : 10
## bruises : 2
## odor : 9
## gill.attachment : 2
## gill.spacing : 2
## gill.size : 2
## gill.color : 12
## stalk.shape : 2
## stalk.root : 5
## stalk.surface.above.ring : 4
## stalk.surface.below.ring : 4
## stalk.color.above.ring : 9
## stalk.color.below.ring : 9
## veil.type : 1
## veil.color : 4
## ring.number : 3
## ring.type : 5
## spore.print.color : 9
## population : 6
## habitat : 7
```

Because we only have one 'veil type' across all of our mushrooms we can disregard this column.

```
df <- (df[-c(17)]) # drop the veil type from the set
```

Here we will turn all of the values from letters to a numerical value

```
# run the as numeric function on each value
df <- data.frame(sapply(df, function (x) as.numeric(x)))
head(df$class, n = 1) # show that class comes out as 1 and 2
```

```
## [1] 2
```

```
df$class <- sapply(df$class, function (x) x - 1) # make class binary
```

2 Modelling / Classification

I will be using the Random Forest model to handle the data as it is likely to have the highest success rate in predict whether or not the mushrooms are edible or not.

2.1 Building Model

2.1.1 Divide The Dataset

Split the dataset with 70% for training and the remainder for testing the model

```
i <- sample(nrow(df), 0.7 * nrow(df)) # get 70% of the elements

train = df[i,] # get all the rows that match i
test =  df[-i,] # get all minus i
```

2.1.2 Building the Model

```
library(randomForest)
set.seed(77) # set the seed for reproducibility

# use randomForest on our training set
forest <- randomForest(as.factor(class) ~ . ,
                           data = train, importance = T, ntree = 50)

pre <- predict(forest, test) # use our model to predict the test set
# create a new set from our prediction
sol <- data.frame(class = test$class, edible = pre)
```

2.2 Testing and Evaluation

```
print(forest)
```

```
##
## Call:
##  randomForest(formula = as.factor(class) ~ ., data = train, importance = T,      ntree = 50)
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##      0    1 class.error
## 0 2964    0           0
## 1    0 2722           0
```

Using all of the features of the dataset we are able to be one hundred percent accurate with our predictions.

```
table(pre == test$class) # show the amount of matches
```

```
##
## TRUE
## 2438
```

Quickly testing whether our test dataset matches our prediction of the test dataset, which it does.

```
plot(forest, main = 'Error Rate For Initial Model', frame = F, col = 'blue')
```

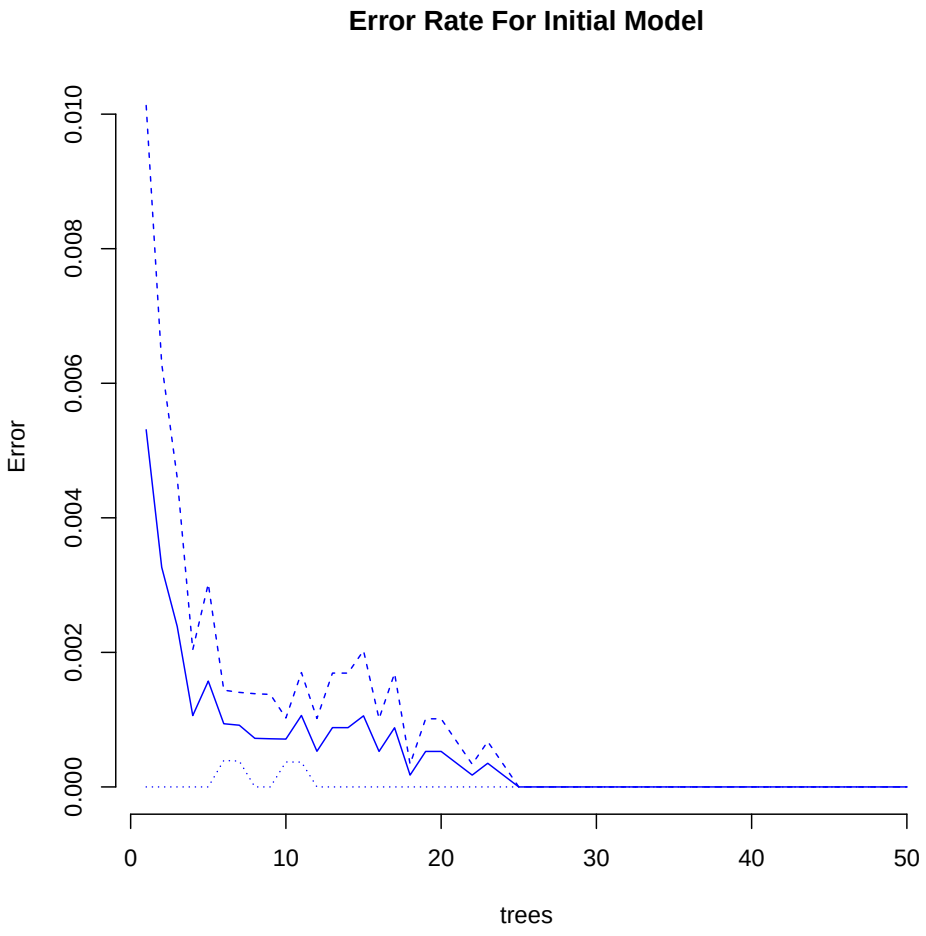

Figure 4: Error rate for the initial random forest model using all features

The graph shows us that it takes approximately thirty tree's before the model is able to be a hundred percent accurate.

2.2.1 Importance

```
varImpPlot(forest, main = 'Importance Rating by Feature', frame = F)
```

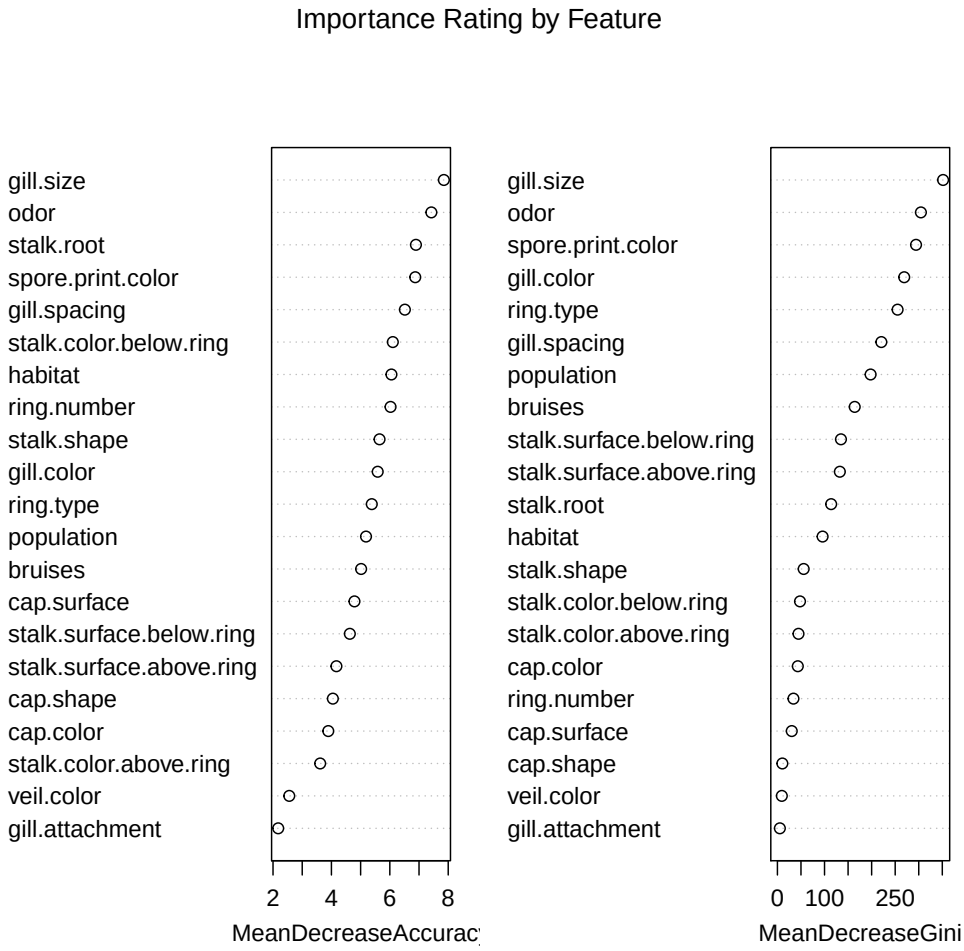

Figure 5: Importance Rating For Each Feature

This graph shows the importance of each of the features when predicting the edibility of the mushroom. From this graph we can see a number of features are unimportant to the model such as veil.color, gill.attachment, etc. these can effectively be ignored in the model without much or any effect on the accuracy of the model.

2.2.2 Confusion Matrix

```
confusionMatrix(pre, test$class) # gen a confusion matrix on our prediction
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1244    0
##          1    0 1194
##
##                Accuracy : 1
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.5103
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5103
##          Detection Rate : 0.5103
##    Detection Prevalence : 0.5103
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

We can see here that the accuracy is rated at 100% which is supported by what we have seen from our prediction data when compared to the test data. From all of this we can be fairly sure in our assertion that our model is entirely accurate.

3 Improving Performance

3.1 Shrinking The Input Features

By looking at the graph of importance from our original model we can select the most important features

```
# reduce the amount of features we give the randomForest
forSmall <- randomForest(as.factor(class) ~ odor + gill.color + gill.size +
                spore.print.color + ring.type + stalk.root + habitat + population + bruises,
        data = train, ntree = 50)
```

We can see that this new model still maintains the same level of accuracy. It is possible to achieve the same accuracy with one less feature however on some iterations of randomForest it would return with a 0.02% error rate

```
print(forSmall) # print out the forest so we can see the accuracy
```

```
##
## Call:
##  randomForest(formula = as.factor(class) ~ odor + gill.color +     gill.size + spore.print.color + ring.type
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##      0    1 class.error
## 0 2964    0           0
## 1    0 2722           0
```

And if we test this model in a prediction and compare that with our test dataset

```
table(predict(forSmall, test) == test$class) # check a prediction against the test data
```

```
##
## TRUE
## 2438
```

We can see it does indeed match

```r
plot(forSmall, main = 'Error Rate With Less Features', frame = F, col = 'blue')
```

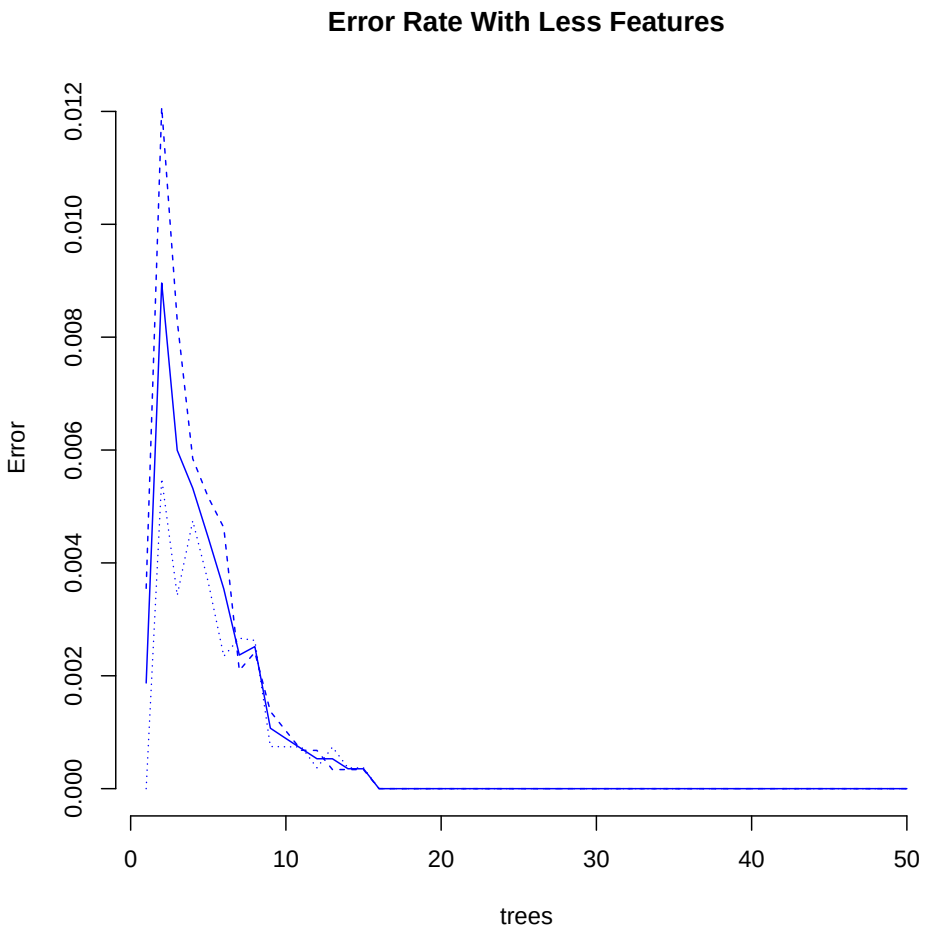

Figure 6: Error rate for random forest model using only the most important features

3.2 Using a Conditional Tree To Compare

3.2.1 Train The Method

```r
ct <- train(as.factor(class) ~ . , data = train, method ='ctree') # train using ctree
```

3.2.2 Predict With The Model

```r
ctpre <- predict(ct, test) # make a predeiction
```

3.2.3 Eveluate the accuracy

```r
confusionMatrix(ctpre, test$class) # compare prediction to the test
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    0    1
##          0 1244    0
##          1    0 1194
##
##                Accuracy : 1
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.5103
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5103
##          Detection Rate : 0.5103
##    Detection Prevalence : 0.5103
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

```
table(ctpre == test$class) # show how many are correct
```

```
##
## TRUE
## 2438
```

This shows us that the the method is not 100% accurate though the margin of error is very small at less than 1%.

3.3 Change Test/Train Scale

For this section I will reduce the training size for our random forest model to see if it still maintains a high level of accuracy. If we see a fall in accuracy then we know that our original method was subject to over fitting and will need tuning.

3.3.1 Build With Redefined Scale

```
ismall <- sample(nrow(df), 0.1 * nrow(df)) # get 10% of the elements

trsmall = df[ismall,]
tesmall = df[-ismall,]
```

```
fsmall <- randomForest(as.factor(class) ~ . ,
                         data = trsmall, importance = T, ntree = 50) # create rf

presmall <- predict(forest, tesmall) # predict

solsmall <- data.frame(class = tesmall$class, edible = presmall) # new table comparing
```

3.3.2 Analyse The Results

```
print(fsmall) # print out info on our rf model
```

```
##
## Call:
##  randomForest(formula = as.factor(class) ~ ., data = trsmall,     importance = T, ntree = 50)
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##     0   1 class.error
## 0 434   0           0
## 1   0 378           0
```

```
table(presmall == tesmall$class) # see how many we got right
```

```
##
## TRUE
## 7312
```

As we can see, the model still retains it's 100% accuracy rating even with only 10% of the the dataset for training. This means that we aren't suffering from any over fitting.

```r
plot(fsmall, main = 'Error Rate For Smaller Training Set', frame = F, col = 'blue')
```

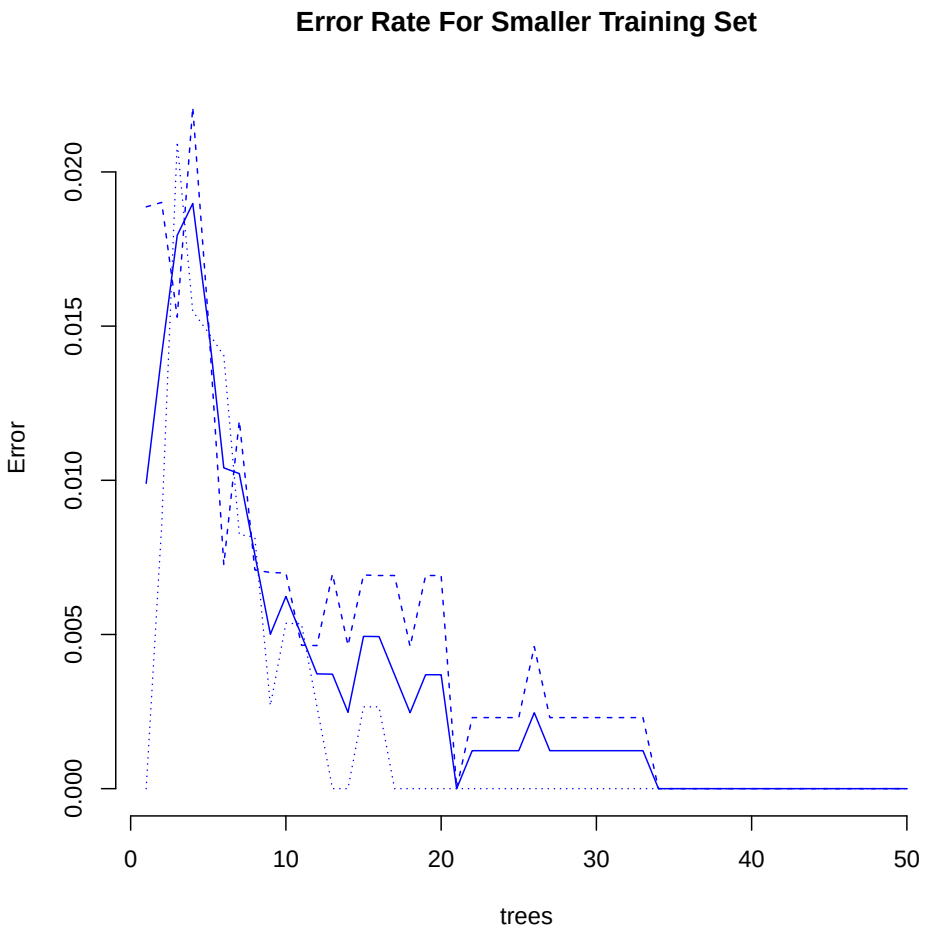

Figure 7: Error rate for random forest model using a smaller training set

Again, we see that around thirty decision tree's are required to hit a 100% success rate which is in line with what we saw when using a 70/30 split.

3.4 Conclusion

The above should show that the original model, using all of the features and a large training set may have been overkill but none the less is still flawless with this dataset. It does not suffer from over fitting or edge cases throwing it out. I believe that random forest was the right choice of model for this dataset, as it is able to reliably be 100% accurate even while restricting the inputs somewhat and it beats the correlation tree by comparison, and though the margin here is small, since we are dealing with potentially serious or fatal poisons here the 0.02% is pretty undesirable.