Code

# Predicting whether a Mushroom is Edible or Poisonous

*MATH 2319 Machine Learning Applied Project Phase II*

*Wesley Paul Nderi (s3635870)*

*11/06/2018*

## 1. Introduction

The objective of this project was to build classifiers to predict whether a mushroom is edible or poisonous based on its descriptive features. The dataset was sourced from Kaggle (https://www.kaggle.com/uciml/mushroom-classification) although it was originally cited in the UCI Machine Learning Repository.

In **Phase I**, we preprocessed the data and omitted some descriptive features. In **Phase II** we shall build three binary-classifiers on the cleaned data. Specifically, the classifiers considered in this report are the Decision tree, Naive-Bayes and Random-Forest.

The rest of this report is organised as follows. Section 2 describes an overview of our methodology. Section 3 discusses the classifiers' fine-tuning process and detailed performance analysis of each classifier.In Section 4, we compare the performance of the classifiers using the same resampling method. Section 5 critiques our methodology. The final section ends with a summary.

## 2. Methodology

We have considered three classifiers - **Decision Tree**, **Naive Bayes** and **Random Forest**. The key consideration for selecting these classifiers was that all the features in the *Mushroom* dataset are categorical in nature and these classifiers would be the most appropriate(Kelleher, Mac Namee & D'Arcy 2015).

Each classifier was trained to make probability predictions in order to allow adjustment of the prediction threshold to refine the performance. The full data set was split into 70 % training set and 30 % test set. In the fine-tuning process, we ran a five-fold cross-validation stratified sampling on each classifier. We employ the use of the stratified sampling in order to address the slight imbalance in the class of the target feature. In addition, as an alternative strategy we also employ a random search with a maximum of 100 iterations to determine the optimum hyperparameters and visualize their importance on the mean of the mean misclassification error rate. We compare and visualize the effect of these hyperparameter values on the mean misclassification error rate.

For each classifer, we determined the optimal probability threshold. Using the tuned hyperparameters and the optimal thresholds, we made predictions on the test data. The performance measure relied on in this report is the **mean misclassification error rate (mmce)**. This is a measure of the number of incorrect predictions made by the model divided by the total number of predictions made.

In addition to mmce, we have also used the confusion matrix and visualizations of the receiver operating characteristic(ROC) curves on the test data to evaluate classifiers' performance. The modelling was implemented in R with the mlr package (Bischl et al. 2016).

## Preliminaries

In this project we used the following R packages.

Hide

```
library(mlr)
library(rpart)
library(rpart.plot)
library(caTools)
library(randomForest)
library(tidyverse)
```

We revisit a quick summary of the data before we embark on modelling.

Hide

```
summary(mushroom1)
```

```
 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
      odor      gill_attachment gill_spacing gill_size
 n      :3528   a: 210          c:6812       b:5612
 f      :2160   f:7914          w:1312       n:2512
 s      : 576
 y      : 576
 a      : 400
 l      : 400
 (Other): 484
   gill_color   stalk_shape stalk_surface_above_ring
 b      :1728   e:3516      f: 552
 p      :1492   t:4608      k:2372
 w      :1202               s:5176
 n      :1048               y:  24
 g      : 752
 h      : 732
 (Other):1170
 stalk_surface_below_ring stalk_color_above_ring
 f: 600                   w      :4464
 k:2304                   p      :1872
 s:4936                   g      : 576
 y: 284                   n      : 448
                          b      : 432
                          o      : 192
                          (Other): 140
 stalk_color_below_ring veil_color ring_number ring_type
 w      :4384           n:  96     n:  36      e:2776
 p      :1872           o:  96     o:7488      f:  48
 g      : 576           w:7924     t: 600      l:1296
 n      : 512           y:   8                 n:  36
 b      : 432                                  p:3968
 o      : 192
 (Other): 156
 spore_print_color population habitat
 w      :2388      a: 384     d:3148
 n      :1968      c: 340     g:2148
 k      :1872      n: 400     l: 832
 h      :1632      s:1248     m: 292
 r      :  72      v:4040     p:1144
 b      :  48      y:1712     u: 368
 (Other): 144                 w: 192
```

There are a total of 21 categorical variables inclusive of the target-**class**.

Hide

```
str(mushroom1)
```

```
'data.frame':   8124 obs. of  21 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_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_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 ...
```

Specifically, we can see that we have a binary target feature **"class"** with the labels *e*-edible and *p*-poisonous.

Hide

```
unique(mushroom1$class)
```

```
[1] p e
Levels: e p
```

In addition we see that there is a slight imbalance between the two levels of the target feature.

Hide

```
table(mushroom1$class)
```

```

   e    p
4208 3916
```

Hide

```
# Set a common random seed for reproducibility
set.seed(123)
```

We split the dataset into 70 % training set and 30 % test set.

Hide

```
split = sample.split(mushroom1$class, SplitRatio = 0.7)
training_setm = subset(mushroom1, split == TRUE)
test_setm = subset(mushroom1, split == FALSE)
```

Both levels appear to be well balanced and representative of the dataset in both the training and test set.

Hide

```
prop.table(table(training_setm$class))
```

```
        e         p 
0.5180236 0.4819764 
```

Hide

```
prop.table(table(test_setm$class))
```

```
        e         p 
0.5178498 0.4821502 
```

## 2.1. Basic configuration

Hide

```
# Configure classification task
task <- makeClassifTask(data = training_setm, target = 'class', id = 'mushroom')
```

Hide

```
# Configure learners with prediction as probability
learner1 <- makeLearner('classif.rpart', predict.type = 'prob',fix.factors.prediction
 = TRUE) # baseline learner
learner2 <- makeLearner('classif.naiveBayes', predict.type = 'prob')
learner3 <- makeLearner('classif.randomForest', predict.type = 'prob')
```

# 3. Model fine-tuning

## 3.1 Decision-tree

The decision tree has several default hyperparameters as can be observed below.

Hide

```
getParamSet(learner1)
```

```
                  Type len  Def   Constr Req Tunable Trafo
minsplit       integer   -   20 1 to Inf   -    TRUE     -
minbucket      integer   -    - 1 to Inf   -    TRUE     -
cp             numeric   - 0.01   0 to 1   -    TRUE     -
maxcompete     integer   -    4 0 to Inf   -    TRUE     -
maxsurrogate   integer   -    5 0 to Inf   -    TRUE     -
usesurrogate   discrete   -    2    0,1,2   -    TRUE     -
surrogatestyle discrete   -    0      0,1   -    TRUE     -
maxdepth       integer   -   30  1 to 30   -    TRUE     -
xval           integer   -   10 0 to Inf   -   FALSE     -
parms          untyped   -    -        -   -    TRUE     -
```

Using default parameters for each of the hyperparameters as listed above, we can evaluate the mmce performance using a 5 fold cross validation and we observe that the error, 0.006, is quite small.

Hide

```
set.seed(123)
rdesc2 <- makeResampleDesc("CV", iters = 5,predict = 'both')
r1    <- resample("classif.rpart",task, rdesc2)
```

```
Resampling: cross-validation
Measures:             mmce.test
[Resample] iter 1:    0.0105448
[Resample] iter 2:    0.0017575
[Resample] iter 3:    0.0043975
[Resample] iter 4:    0.0052770
[Resample] iter 5:    0.0087951


Aggregated Result: mmce.test.mean=0.0061544
```

We shall start by creating a search for the most appropriate hyperparameters in order to fine tune the decision tree and possibly have a lower misclassfication error rate.

We shall focus on 3 parameters: *minsplit* which represents the minimum number of observations in a node for a split to take place; *minbucket* represents the minimum number of observations to keep in terminal nodes and *cp* is the complexity parameter. The smaller it is, the more the tree will focus on specific relations in the data which might result in overfitting.

Hide

```
ps1 <- makeParamSet(
makeIntegerParam('maxdepth', lower = 2, upper = 30),
makeNumericParam("cp", lower = 0.001, upper = 0.03)
)
```

# 3.2 Naive Bayes

With the NaiveBayes learner, we obtain the default parameters as shown below.

Hide

```
getParamSet(learner2)
```

```
          Type len Def   Constr Req Tunable Trafo
laplace numeric   -   0 0 to Inf   -    TRUE     -
```

This parameter is used to smooth conditional probabilities for the features.

Using its default value of 0 and a 5 fold cross validation and we observe that the error is about 0.05.

Hide

```
set.seed(123)
rdesc2 <- makeResampleDesc("CV", iters = 5,predict = 'both')
r2    <- resample("classif.naiveBayes",task, rdesc2)
```

```
Resampling: cross-validation
Measures:             mmce.test
[Resample] iter 1:    0.0659051
[Resample] iter 2:    0.0448155
[Resample] iter 3:    0.0501319
[Resample] iter 4:    0.0545295
[Resample] iter 5:    0.0430959


Aggregated Result: mmce.test.mean=0.0516956
```

We shall proceed to fine tune Laplace by creating a search for values from 0 to 50 as shown below.

Hide

```
ps2 <- makeParamSet(
  makeNumericParam('laplace', lower = 0, upper = 50)
)
```

# 3.3 Random Forest

With the randomForest learner, we obtain the default hyperparameters as shown below.

Hide

```
getParamSet(learner3)
```

```
                   Type  len   Def   Constr Req Tunable
ntree           integer    -   500 1 to Inf   -    TRUE
mtry            integer    -     - 1 to Inf   -    TRUE
replace         logical    -  TRUE        -   -    TRUE
classwt     numericvector <NA>     - 0 to Inf   -    TRUE
cutoff      numericvector <NA>     -   0 to 1   -    TRUE
strata          untyped    -     -        -   -   FALSE
sampsize    integervector <NA>     - 1 to Inf   -    TRUE
nodesize        integer    -     1 1 to Inf   -    TRUE
maxnodes        integer    -     - 1 to Inf   -    TRUE
importance      logical    - FALSE        -   -    TRUE
localImp        logical    - FALSE        -   -    TRUE
proximity       logical    - FALSE        -   -   FALSE
oob.prox        logical    -     -        -   Y   FALSE
norm.votes      logical    -  TRUE        -   -   FALSE
do.trace        logical    - FALSE        -   -   FALSE
keep.forest     logical    -  TRUE        -   -   FALSE
keep.inbag      logical    - FALSE        -   -   FALSE
            Trafo
ntree           -
mtry            -
replace         -
classwt         -
cutoff          -
strata          -
sampsize        -
nodesize        -
maxnodes        -
importance      -
localImp        -
proximity       -
oob.prox        -
norm.votes      -
do.trace        -
keep.forest     -
keep.inbag      -
```

Using these default values and a 5 fold cross validation and we observe that the error is 0.

Hide

```
set.seed(123)
rdesc2 <- makeResampleDesc("CV", iters = 5,predict = 'both')
r3    <- resample("classif.randomForest",task, rdesc2)
```

```
Resampling: cross-validation
Measures:             mmce.test
[Resample] iter 1:    0.0000000
[Resample] iter 2:    0.0000000
[Resample] iter 3:    0.0000000
[Resample] iter 4:    0.0000000
[Resample] iter 5:    0.0000000


Aggregated Result: mmce.test.mean=0.0000000
```

Although this cannot be improved any further, we can attempt to fine-tune *mtry* i.e mumber of variables randomly sampled as candidates at each split. It is suggested in Breiman (2001) that in a classification problem, the optimum value of mtry = √p where p is the number of descriptive features. In our case, √p = √20 = 4.47. We therefore experimented with mtry = 3, 4, and 5 to see if tuning it keeps the error just as low.

Hide

```
ps3 <- makeParamSet(
  makeDiscreteParam('mtry', values = c(3,4,5))
)
```

As mentioned above, we shall configure a tune control search through the different ranges of parameters set above and employ a 5-CV stratified sampling strategy.

Hide

```
# Configure tune control
ctrl  <- makeTuneControlGrid()
ctrrandom<-makeTuneControlRandom(maxit = 100L)
rdesc <- makeResampleDesc("CV", iters = 5L, stratify = TRUE)
```

Hide

```
# Configure tune wrapper with tuning settings
tunedLearner1 <- makeTuneWrapper(learner1, rdesc, mmce, ps1, ctrl)
tunedLearner2 <- makeTuneWrapper(learner2, rdesc, mmce, ps2, ctrl)
tunedLearner3 <- makeTuneWrapper(learner3, rdesc, mmce, ps3, ctrl)
```

As an alternative approach, we can also tell R to pick a random value within the grid search that returns the best MMCE result.

Hide

```
# Using random search to find best parameters for decision tree
set.seed(123)
res_dec = tuneParams('classif.rpart',task,rdesc,measures = mmce,ps1,ctrrandom,show.in
fo = FALSE)
print(res_dec)
```

```
Tune result:
Op. pars: maxdepth=13; cp=0.0014
mmce.test.mean=0.0010551
```

The random search shows that the max depth is 13, cp is 0.0014 in order to yield a mmce mean of 0.001.

Hide

```
# Using random search to find best parameters for NaiveBayes
set.seed(123)
res_nb = tuneParams('classif.naiveBayes', task, rdesc,measures = mmce, ps2,ctrrandom,
 show.info = FALSE)
print(res_nb)
```

```
Tune result:
Op. pars: laplace=0.206
mmce.test.mean=0.0223324
```

The random search shows that optimum laplace value is 0.206 in order to yield a mmce mean of 0.02.

Hide

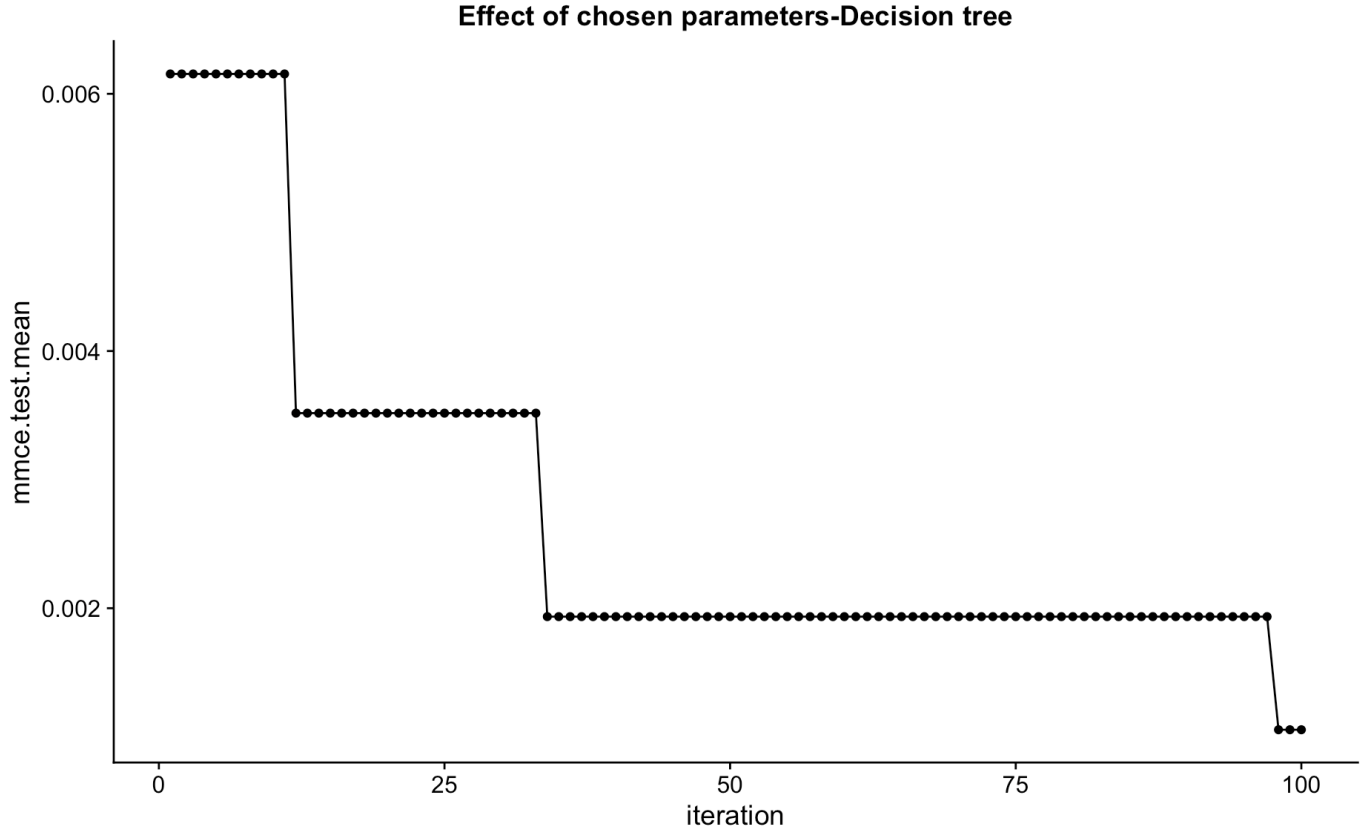
```
# Using random search to find best parameters for RandomForest
set.seed(123)
res_rf = tuneParams('classif.randomForest', task, rdesc,measures = mmce, ps3,ctrrando
m, show.info = FALSE)
print(res_rf)
```

```
Tune result:
Op. pars: mtry=4
mmce.test.mean=0.0000000
```

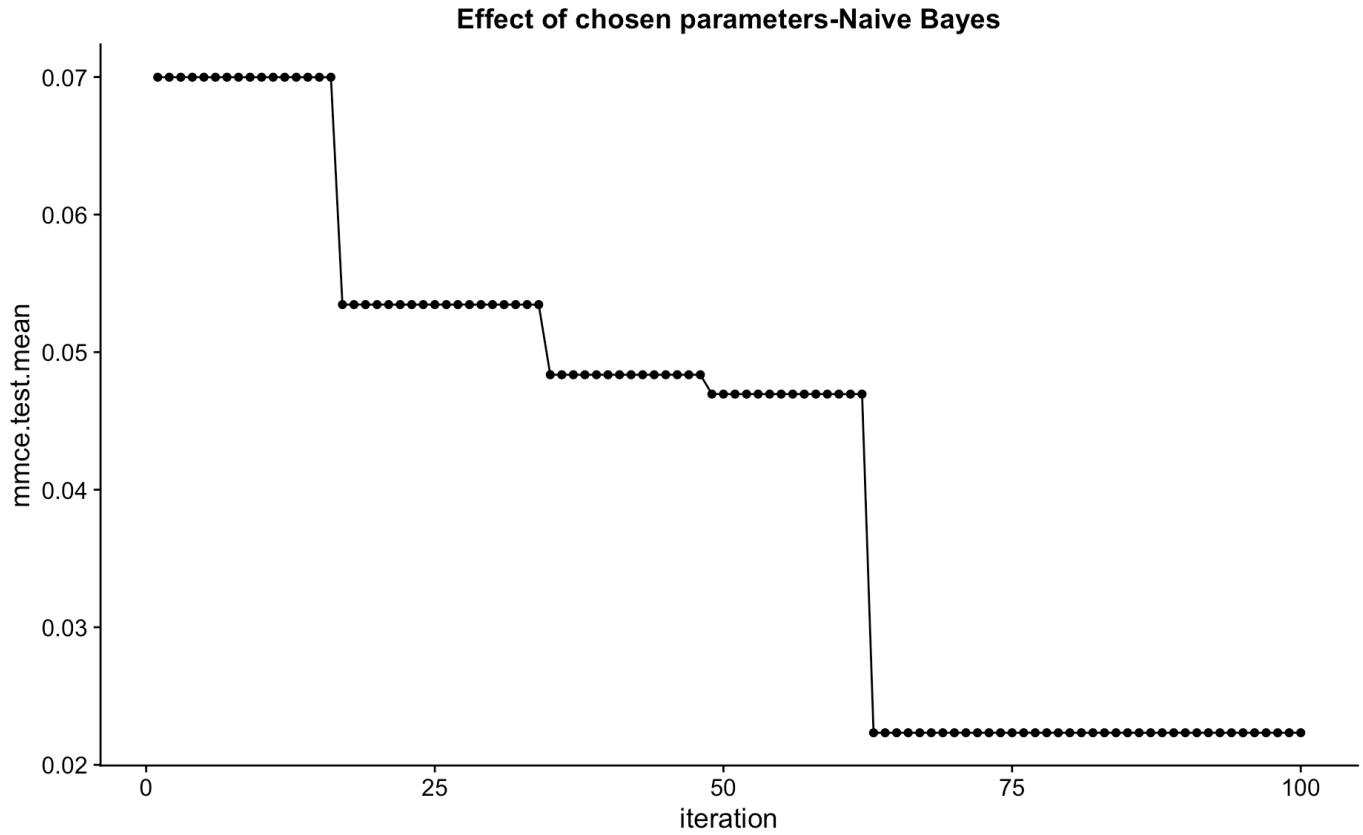
The random search shows that an mtry value of 4 also yields a mmce mean of 0.

# 3.4 Effect of random search hyperparameters on MMCE

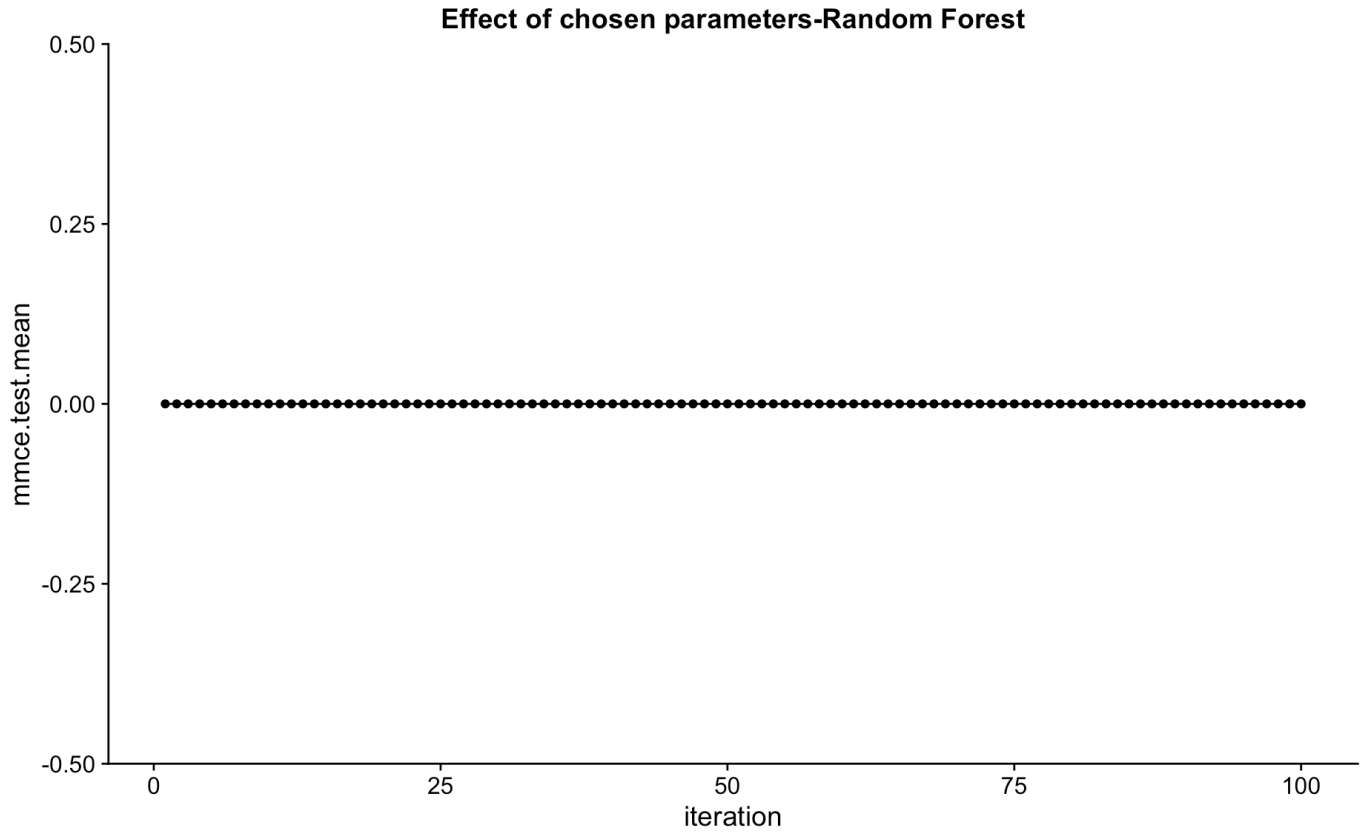
We can visualise the importance of the above hyperparameters on the mean of the mmce rate in each of the classifiers.




These hyperparameters appears to have a considerably significant effect on the mmce.




This smoothing hyperparameter appears to have a considerably significant effect on the mmce and reduces it significantly.




We observe that in the case of the random Forest, the chosen value of **mtry** results in an mmce value of 0 at any iteration.

We can observe the results of the grid search hyperparameters.

The optimum values are given as:

- A decision tree with **maxdepth** of 21 and a cp of 0.001 would result in a mmce.test.mean=0.001.
- A Naive Bayes model with a **laplace** value of 0 would result in a mmce.test.mean=0.05.
- A Random Forest model with **mtry** value of 3 would result in a mmce.test.mean=0.

With these observations in mind, we shall go on to train the data on the tuned learners, observe their threshold values and visualize their probability thresholds.

Hide

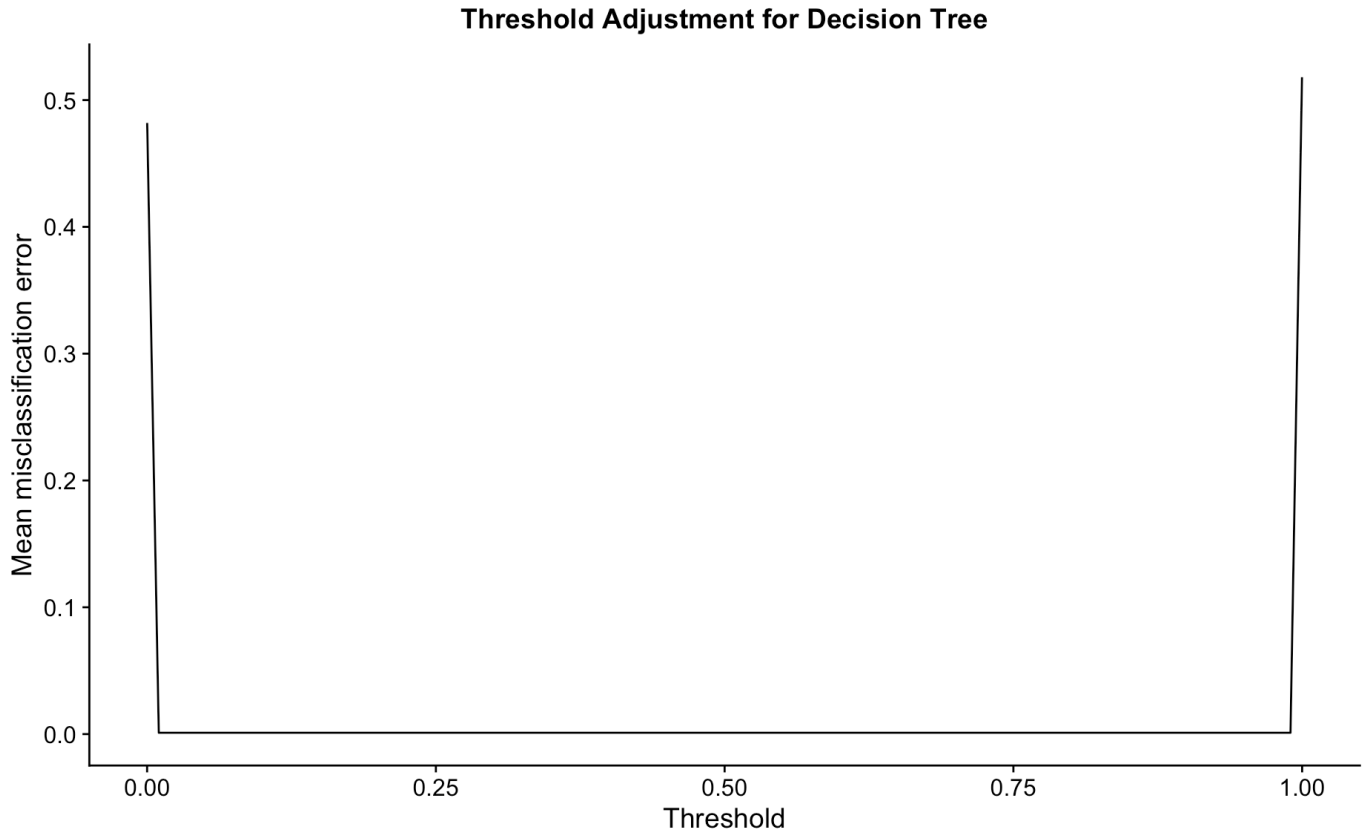
```
# Predict on training data
tunedPred1 <- predict(tunedMod1, task)
tunedPred2 <- predict(tunedMod2, task)
tunedPred3 <- predict(tunedMod3, task)
```

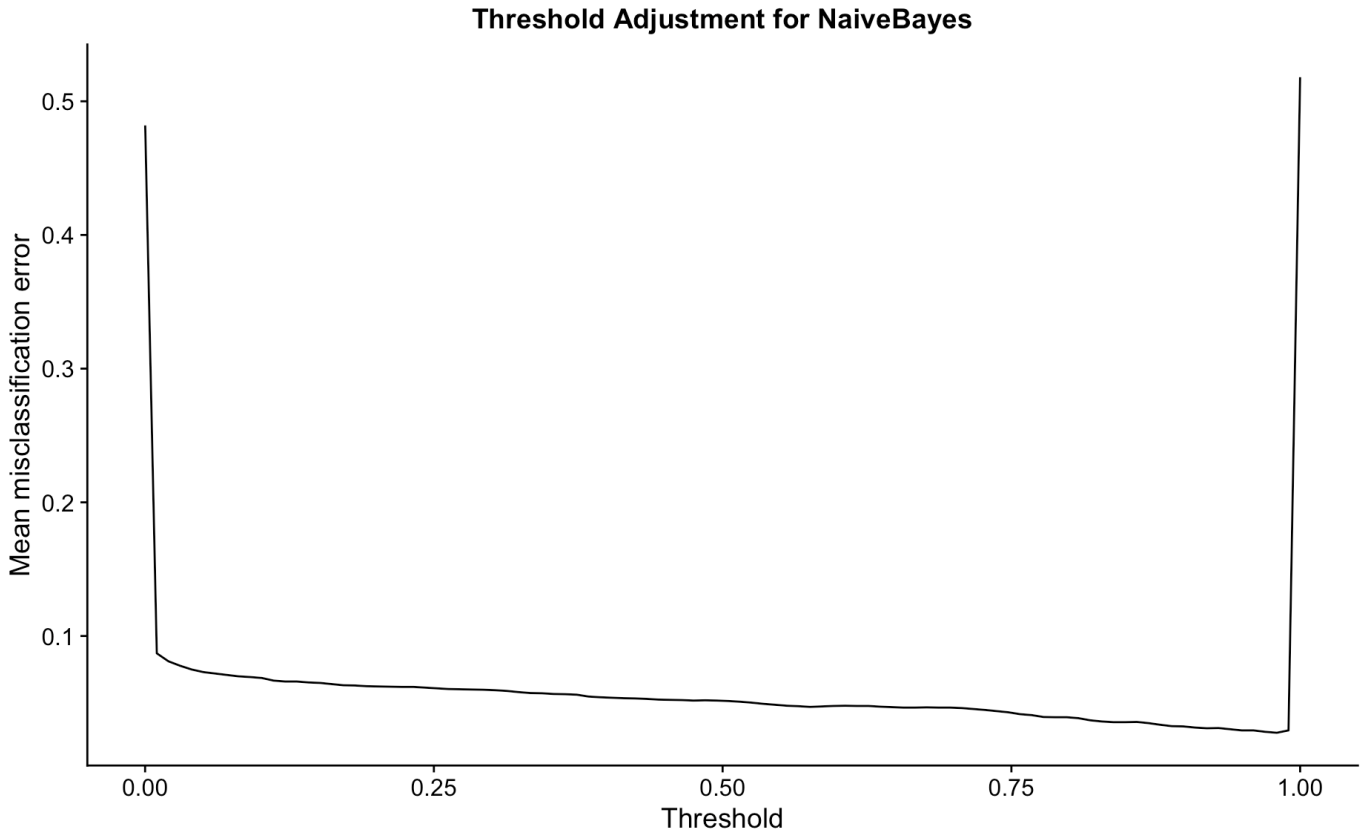
Hide

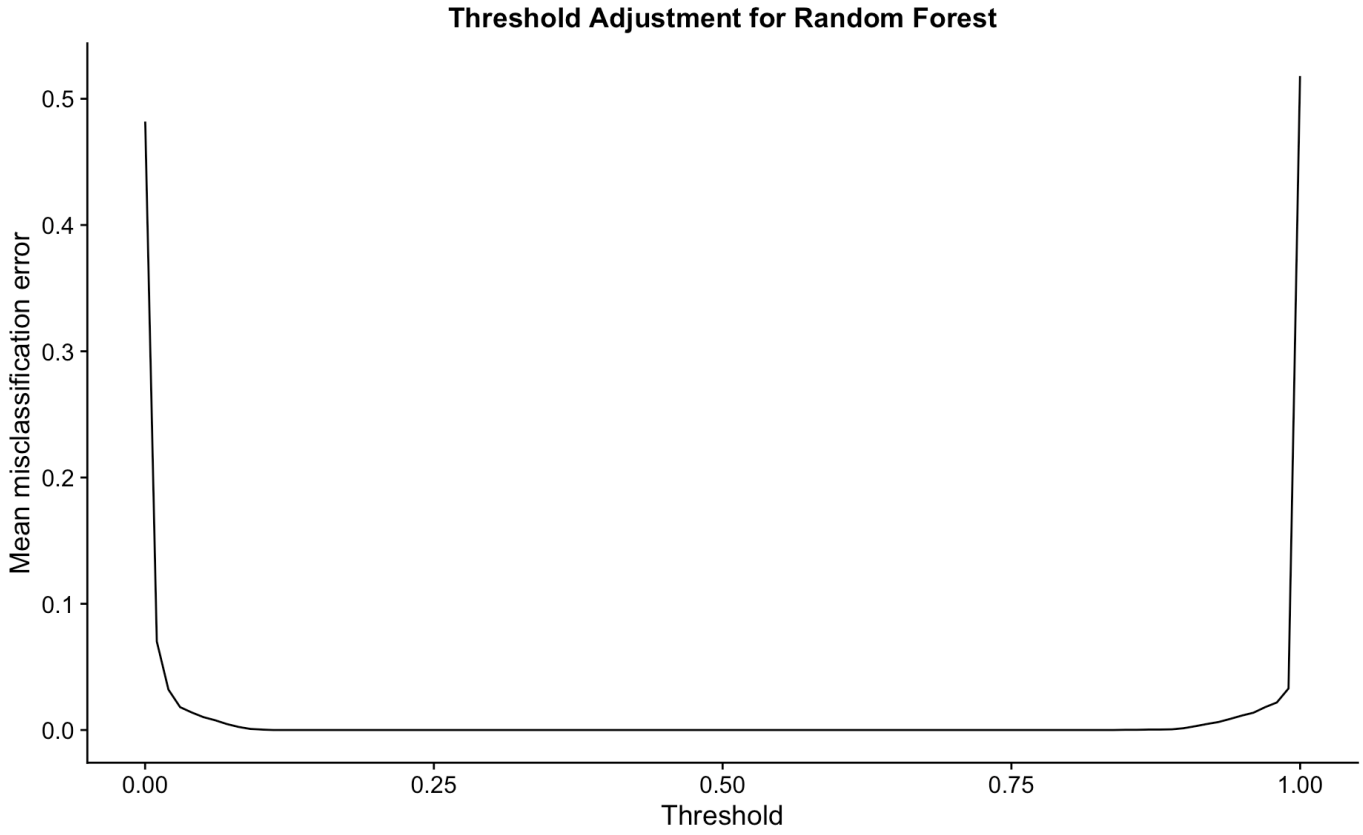
```
# Obtain threshold values for each learner
d1 <- generateThreshVsPerfData(tunedPred1, measures = list(mmce))
d2 <- generateThreshVsPerfData(tunedPred2, measures = list(mmce))
d3 <- generateThreshVsPerfData(tunedPred3, measures = list(mmce))
```

# 3.5 Threshold Adjustment

The following plots depict the value of mmce vs. the range of probability thresholds. The thresholds are approximately **0.01**, **0.98**, and **0.11** for the decision tree, Naive Bayes, and Random Forest classifiers respectively. These thresholds were used to determine the probability of a poisonous mushroom.






# 4. Evaluation

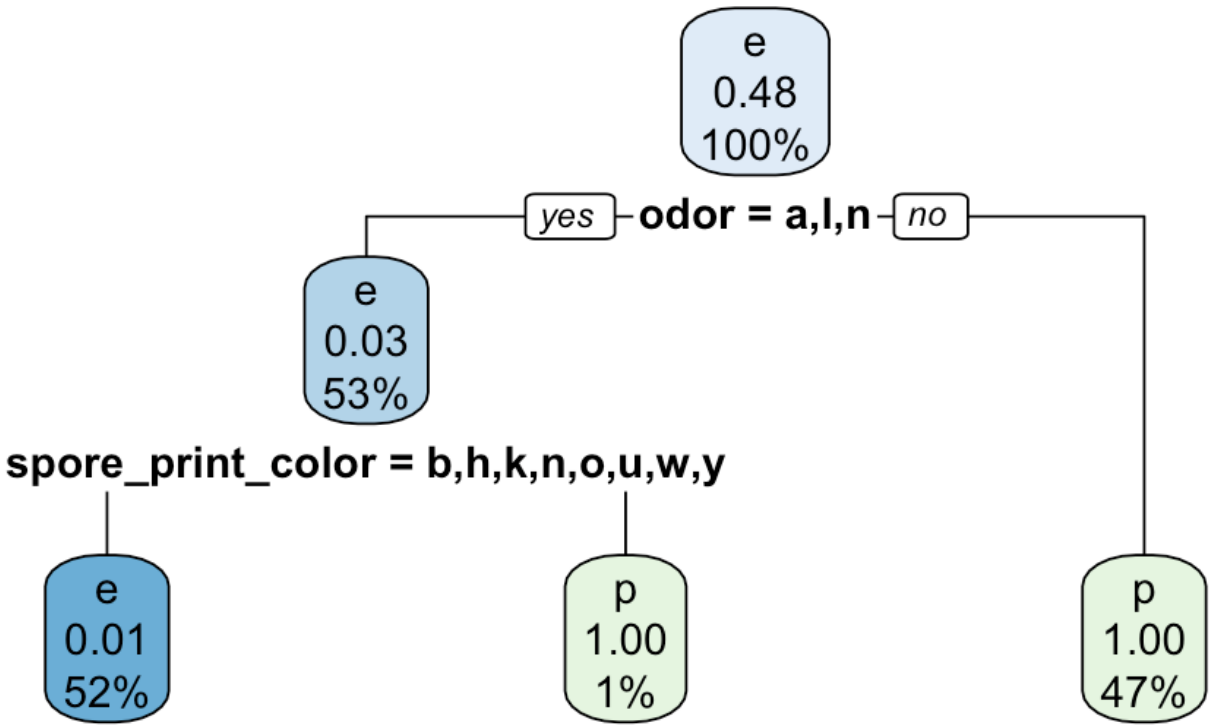
## 4.1 Confusion Matrix

Using the parameters and threshold levels, we calculated the confusion matrix for each classifier. The confusion matrix of the decision tree is shown below:

```
Relative confusion matrix (normalized by row/column):
        predicted
true     e           p           -err.-
  e      1.000/0.998 0.000/0.000 0.000
  p      0.002/0.002 0.998/1.000 0.002
  -err.-       0.002       0.000 8e-04


Absolute confusion matrix:
        predicted
true        e    p -err.-
  e      1262    0      0
  p         2 1173      2
  -err.-    2    0      2
        mmce
0.0008206812
```

The missclassification error rate is calculated as 0.0008. This model does a good job of classifying the edible target value and misclassifies 2 poisonous mushrooms as edible mushrooms.

This can be visualised as shown below. The decision tree does a good job of splitting the dataset on what it considers is the feature with the highest probability,odor.




The confusion matrix for the Naive Bayes is shown below:

```
Relative confusion matrix (normalized by row/column):
        predicted
true     e         p         -err.-
  e      0.97/0.99 0.03/0.03 0.03
  p      0.01/0.01 0.99/0.97 0.01
  -err.-      0.01      0.03 0.02


Absolute confusion matrix:
        predicted
true        e    p -err.-
  e      1227   35     35
  p        17 1158     17
  -err.-   17   35     52
      mmce
0.02133771
```

The missclassification error rate is calculated as 0.02. This classifier generates 35 false negatives and 17 false positives.

The confusion matrix for the Random Forest model is shown below:

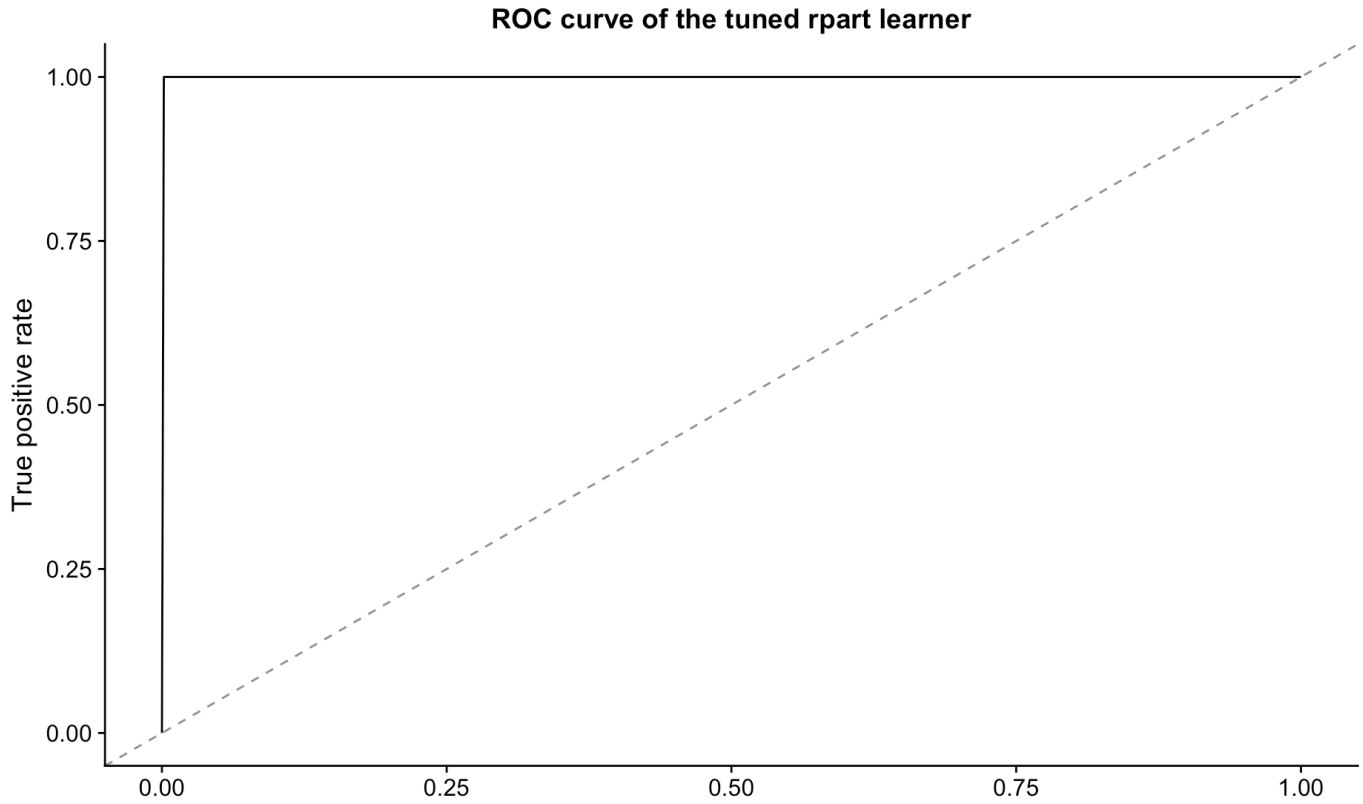
```
Relative confusion matrix (normalized by row/column):
        predicted
true     e           p           -err.-
  e      1e+00/1e+00 0e+00/0e+00 0e+00
  p      9e-04/8e-04 1e+00/1e+00 9e-04
  -err.-       8e-04       0e+00 4e-04


Absolute confusion matrix:
        predicted
true        e    p -err.-
  e      1262    0      0
  p         1 1174      1
  -err.-    1    0      1
        mmce
0.0004103406
```

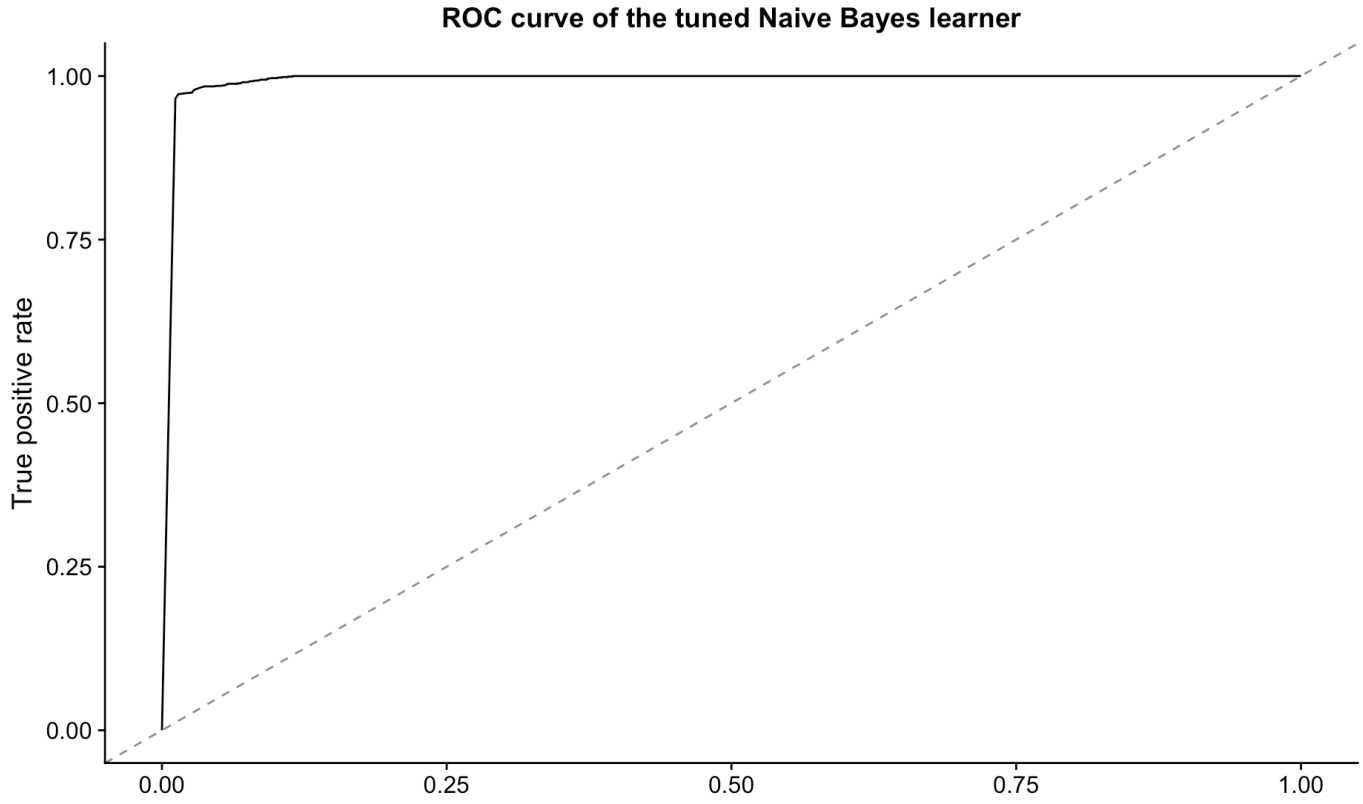
The missclassification error rate is calculated as 0.0004.This model generates 1 false positive and no false negatives.

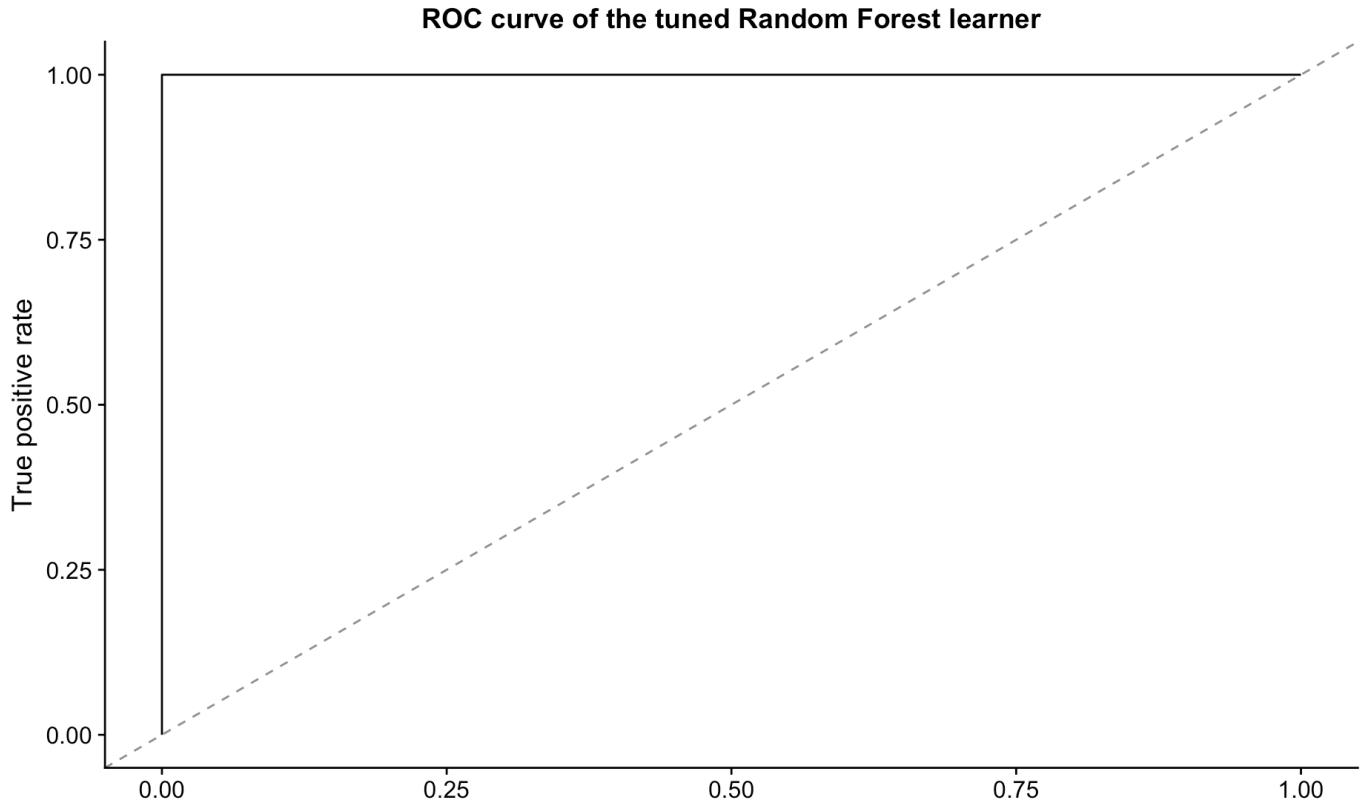
All classifiers accurately distinguish between edible and poisonous mushrooms with very little error observed. The decision tree and random forest models stand out only missclassifying 2 and 1 poisonous mushrooms as edible(false positives). The Naive Bayes model does not perform as well and has 17 false positives and 35 false negatives with an error of 0.02. Based on class accuracy and mmce, we conclude that the random forest classifier is the better model.

# 4.2 ROC curves

The ROC curve is a visual representation of the true positive rate on the vertical axis and false positive rate on the horizontal axis at a given threshold. This curve gives a visual indication of the strength of the model. A line along the diagonal is representative of a model that makes random predictions e.g a coin flip. The closer the curve is to the top left, the more predictive the model(Kelleher, Mac Namee & D'Arcy 2015).


ROC curve of the tuned rpart learner
True positive rate
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00


ROC curve of the tuned Naive Bayes learner
True positive rate
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00




The three classifiers all appear to be very strong and have a relatively high area under the curve(AUC).

# 5. Discussion

The previous section showed that all classifiers considerably well in predicting whether a mushroom was poisonous or edible. The random forest and decision tree are much better then the Naive bayes classifier and have considerably less false positives and false negatives. The slight class imbalance does not seem to affect the models. The models are clearly suitable to work on this classification task with all features categorical.

Despite the much higher accuracy in predictions in the decision tree and random forest, these models also have a higher tendency to overfit.

In addition, the Naive Bayes model assumes the descriptive features are normally distributed which is not always the case.

It is also important to consider that the random forest classifier is at an advantage because it is able to run multiple bagged models at each iteration of 500 trees by default.

# 6. Conclusion

The data was split into a training set or test set and validated using a 5 fold cross validation technique. We perform a comparison of the mean misclassification error rate before and after tuning the hyperparameters as well as explore the effect of these on the mmce. Among the three classifiers used in this report, the Random Forest outperforms the decision tree and Naive Bayes models. The random forest only marginally outperforms the decision tree although both have a strong tendency to overfit. In general, the models all perform considerably well in distinguishing between edible and poisonous mushrooms.

# References

Kelleher J, Mac Namee B & D'Arcy A 2015, *Fundamentals of Machine Learning for Predictive Analytics: Algorithms, Worked Examples and Case Studies*, Cambridge, Massachusetts Institute of Technology.

Svetnik V, Liaw A, Tong C & Wang T 2004, 'Application of Breiman's Random Forest to Modeling Structure-Activity Relationships of Pharmaceutical Molecules', *Lecture Notes in Computer Science*, vol. 3077.