Data Mining and Statistical Learning

HW5

Introduction

In machine learning and data mining, ensemble methods is a technique that combines a myriad of models in order to produce that one final model that can make the best or most accurate prediction– average prediction of all models. By averaging the predictions from many models, the ensemble method can improve their predictive performance.

Ensemble methods employ several techniques like Bayesian model averaging, Stacking, Bagging (Bootstrap aggregating), Random Forest, Boosting, etc. Bagging and the Random forest mainly work on the tree-based method and each tree uses the bootstrap sample from the same training data. Then we'll consider the average of all predictions from many different trees. Meanwhile, boosting uses the key idea of our real weight of the training data Instead of using the re-sampling. So that we will put more weight if we made mis-classifications and then iterate this until we do a good job on the predictions of the training data.

The goal of this homework is to apply random forest and boosting algorithms to a mushroom dataset obtained from the UCI Machine Learning Archive. The goal of this analysis is to classify mushroom (edible vs poisonous) to detect their edibility. The dataset will also be used in the course project.

Problem Statement and Data set

The objective of this analysis is to predict if a mushroom is edible or poisonous based on some key attributes.

The dataset that will be used in this assignment is obtained from the UCI Machine Learning Archive. The dataset consists of **61069** hypothetical mushrooms with caps based on **173** species (**353** mushrooms per species). Each mushroom is classified as edible, poisonous. The dataset consists of 20

attributes where 17 are categorical and 3 are metrical. All of the predictors in the dataset are summarized below.

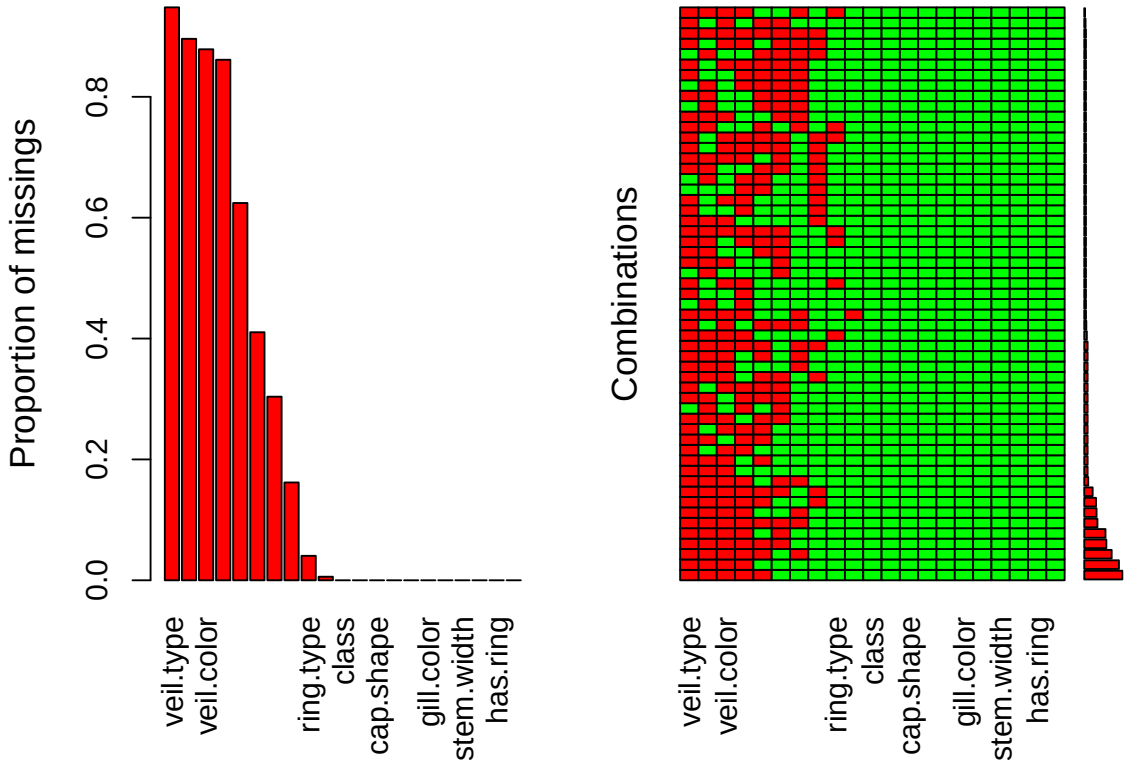
1. cap-diameter (m): float number in cm
2. cap-shape (n): bell=b, conical=c, convex=x, flat=f,sunken=s, spherical=p, others=o
3. cap-surface (n): fibrous=i, grooves=g, scaly=y, smooth=s,shiny=h, leathery=l, silky=k, sticky=t,wrinkled=w, fleshy=e
4. cap-color (n): brown=n, buff=b, gray=g, green=r, pink=p,purple=u, red=e, white=w, yellow=y, blue=l,orange=o, black=k
5. does-bruise-bleed (n): bruises-or-bleeding=t,no=f
6. gill-attachment (n): adnate=a, adnexed=x, decurrent=d, free=e, sinuate=s, pores=p, none=f, unknown=?
7. gill-spacing (n):close=c, distant=d, none=f
8. gill-color (n):see cap-color + none=f
9. stem-height (m):float number in cm
10. stem-width (m):float number in mm
11. stem-root (n):bulbous=b, swollen=s, club=c, cup=u, equal=e,rhizomorphs=z, rooted=r
12. stem-surface (n):see cap-surface + none=f
13. stem-color (n):see cap-color + none=f
14. veil-type (n):partial=p, universal=u
15. veil-color (n):see cap-color + none=f
16. has-ring (n): ring=t, none=f
17. ring-type (n):cobwebby=c, evanescent=e, flaring=r, grooved=g,large=l, pendant=p, sheathing=s, zone=z, scaly=y, movable=m, none=f, unknown=?
18. spore-print-color (n):see cap color
19. habitat (n):grasses=g, leaves=l, meadows=m, paths=p, heaths=h,urban=u, waste=w, woods=d
20. season (n): spring=s, summer=u, autumn=a, winter=w

Exploratory Data Analysis

For any real dataset, we first need to conduct empirical data analysis to have a better understanding of the data. However, our dataset is large with several categorical attributes, so the EDA will be limited.

We will begin EDA by first converting the variables that are in cm to mm for unit standardization purposes. We will also use the full name of each categorical variable for plotting purposes.

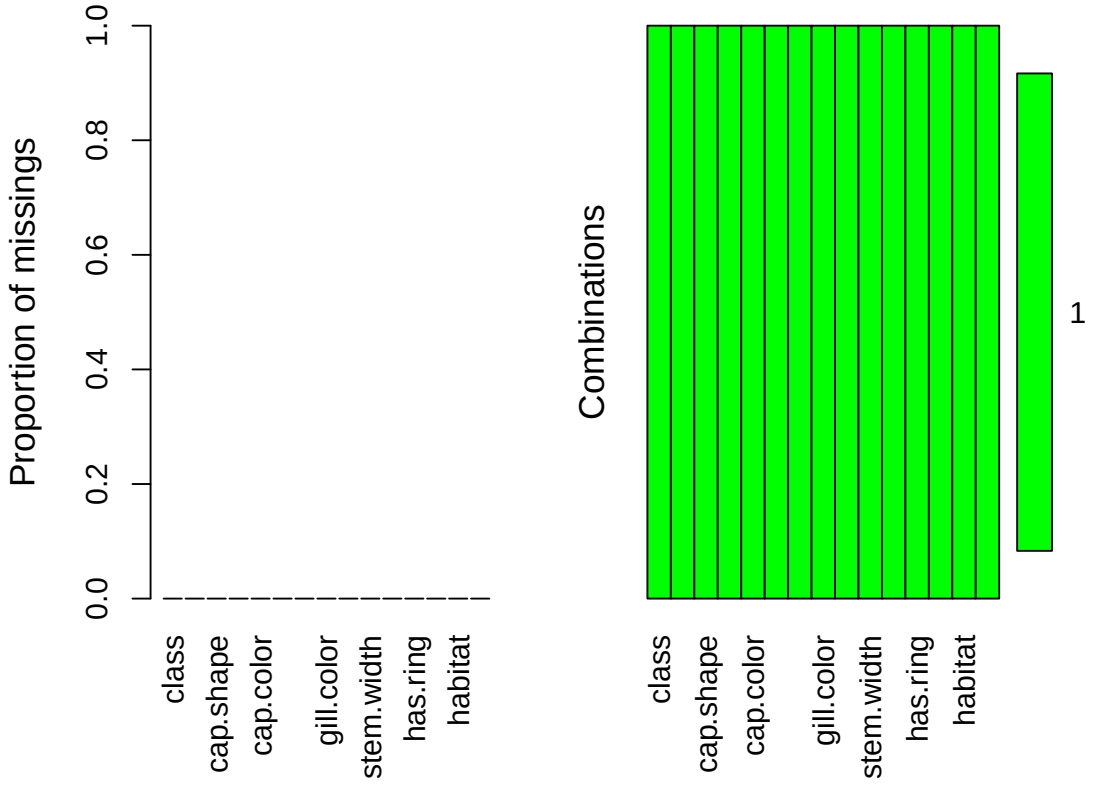
Unfortunately, the dataset contains a lot of missing values (shown below) and needed to be handled




```
##
##  Variables sorted by number of missings:
##               Variable       Count
##             veil.type 0.947976879
##     spore.print.color 0.895953757
##            veil.color 0.878612717
##             stem.root 0.861271676
##          stem.surface 0.624277457
##          gill.spacing 0.410404624
##           cap.surface 0.303787519
##       gill.attachment 0.161849711
```

```
##             ring.type 0.040462428
##               habitat 0.005894971
##                 class 0.000000000
##          cap.diameter 0.000000000
##             cap.shape 0.000000000
##             cap.color 0.000000000
##  does.bruise.or.bleed 0.000000000
##            gill.color 0.000000000
##           stem.height 0.000000000
##            stem.width 0.000000000
##            stem.color 0.000000000
##              has.ring 0.000000000
##                season 0.000000000
```

We will drop any column that has more **65%** missing values, and replace the remaining missing values in categorical variables with **"unknown"**. We inspect the dataset again after pre-processing and missing values imputation (data is clean)




```
##
```

```
##  Variables sorted by number of missings:
##              Variable Count
##                 class     0
##          cap.diameter     0
##             cap.shape     0
##           cap.surface     0
##             cap.color     0
##  does.bruise.or.bleed     0
##       gill.attachment     0
##            gill.color     0
##           stem.height     0
##            stem.width     0
##            stem.color     0
##              has.ring     0
##             ring.type     0
##               habitat     0
##                season     0
```

After imputing missing values, we look at the distribution of the three continous variables in the dataset. We see a high correlation between stem width and cap diameter especially for poisonous mushrooms. This indicate that we may need to drop either stem width or cap diameter from the dataset.

To have a better understanding of how we can diffentiate between poisonous and edible mushrooms, we will plot couple of the features. Below, we see how poisonous/edible mushroom differ with respect to cap surface type and the cap color of a mushroom.

We see that mushrooms with gray cap color and grooves cap surface are most likely safe to eat, while those with a pink cap color are poisonous.

Here again, most mushrooms with a bell cap shape are probably not safe to eat especially those that are green, pink, purple, blue, orange, or red.

Here, we look at mushrooms by habitat. We clearly see that most mushroom in urban or waste areas are edible (I'd probably not eat a mushroom growing near waste lands)

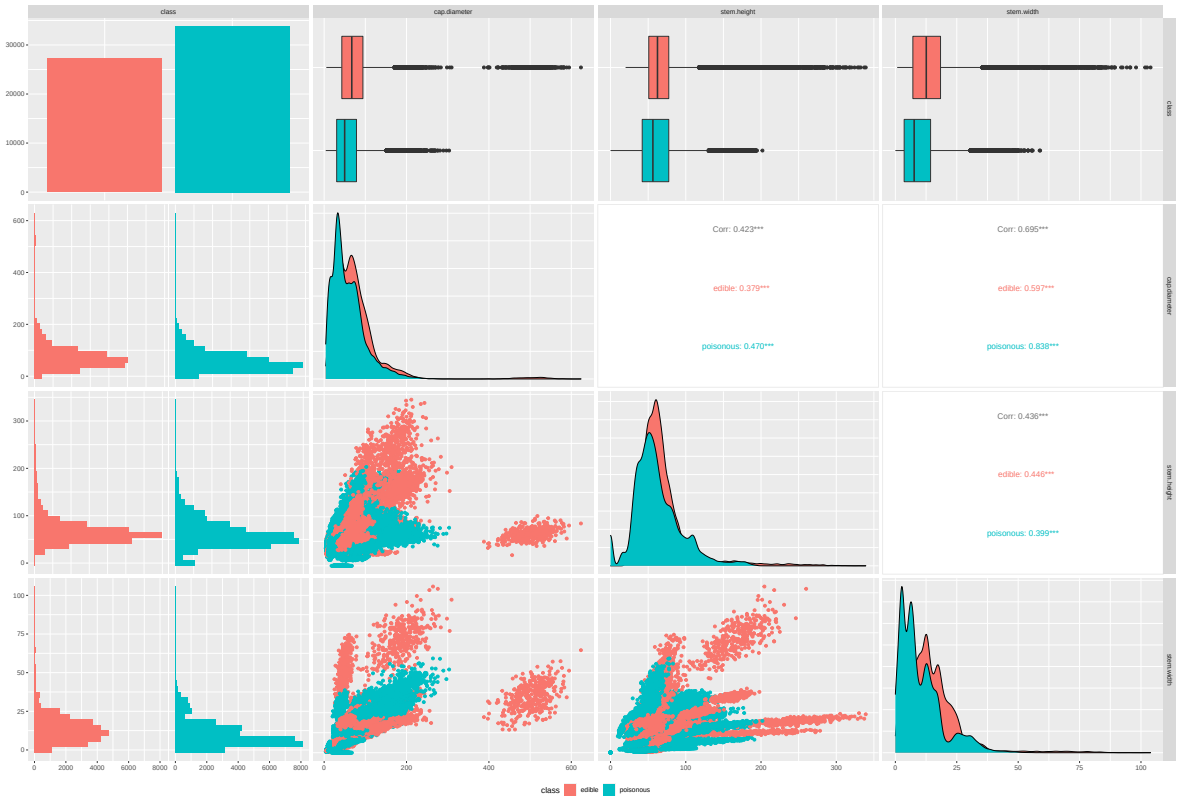


Figure 1: Pairs plot of the continous variables in the dataset

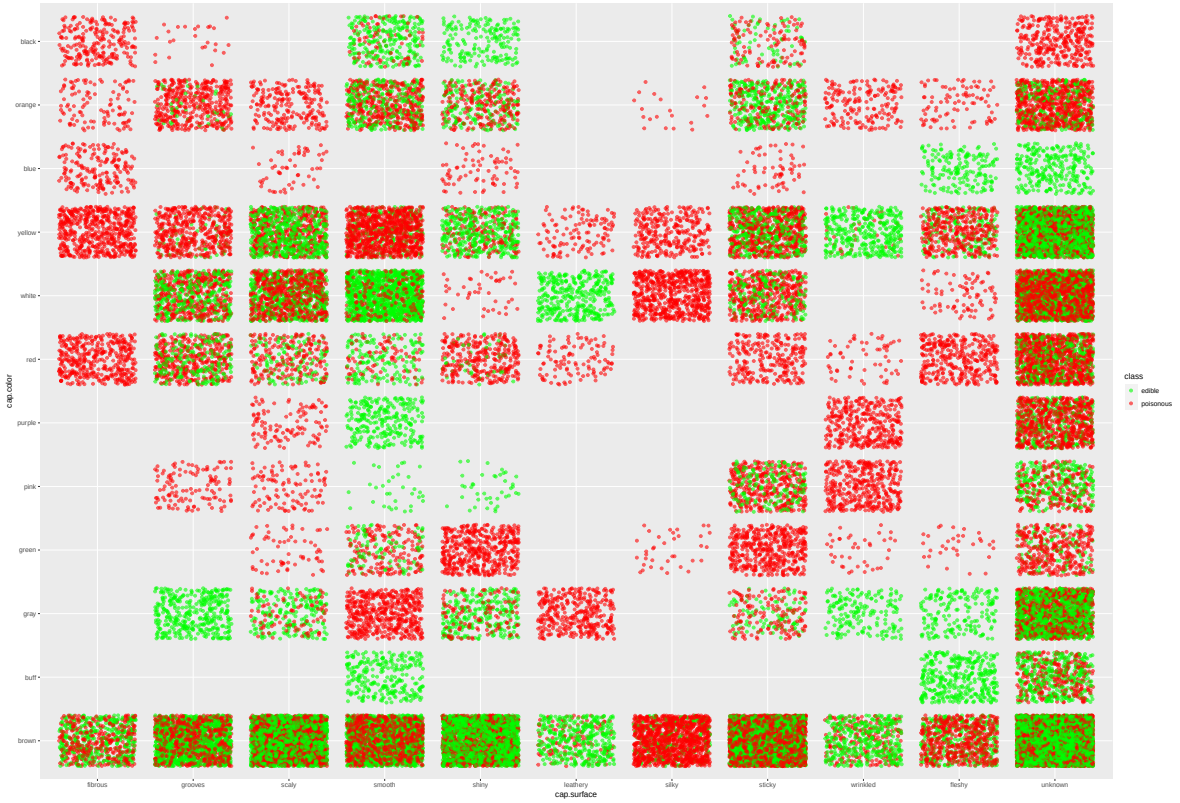


Figure 2: Mushroom cap color vs cap surface type

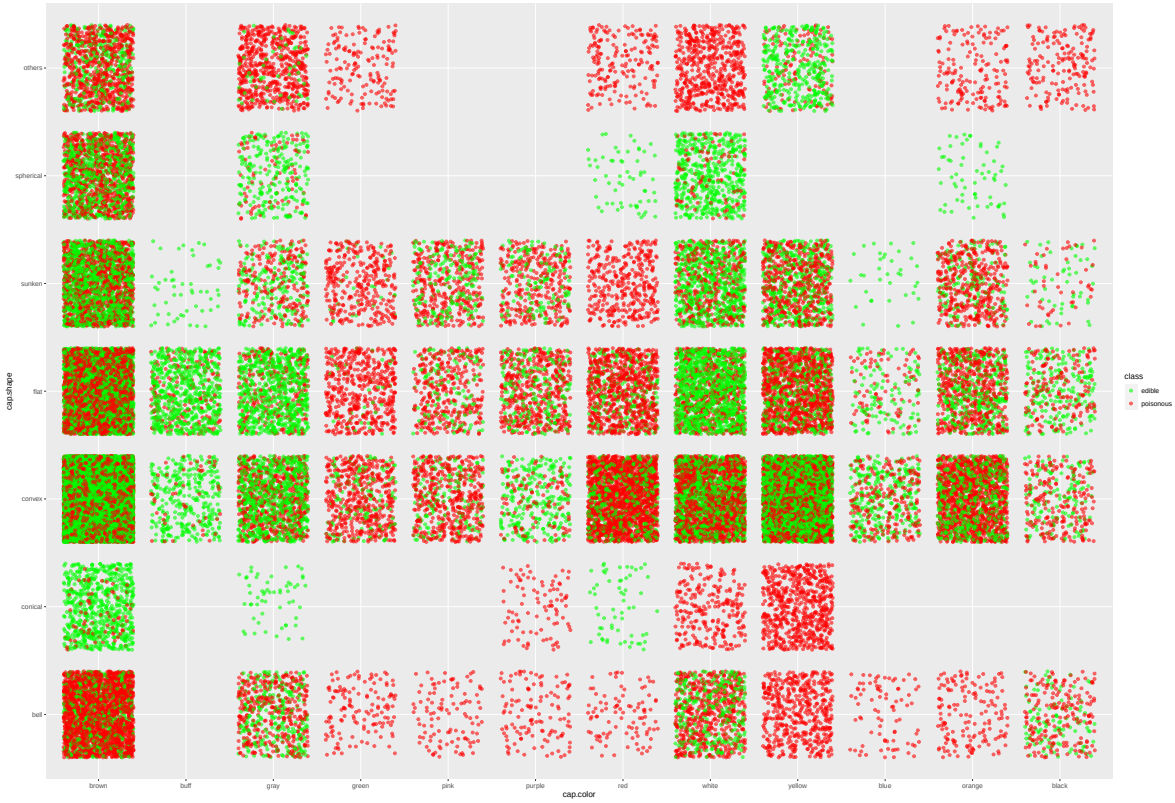


Figure 3: Mushroom cap color vs cap shape

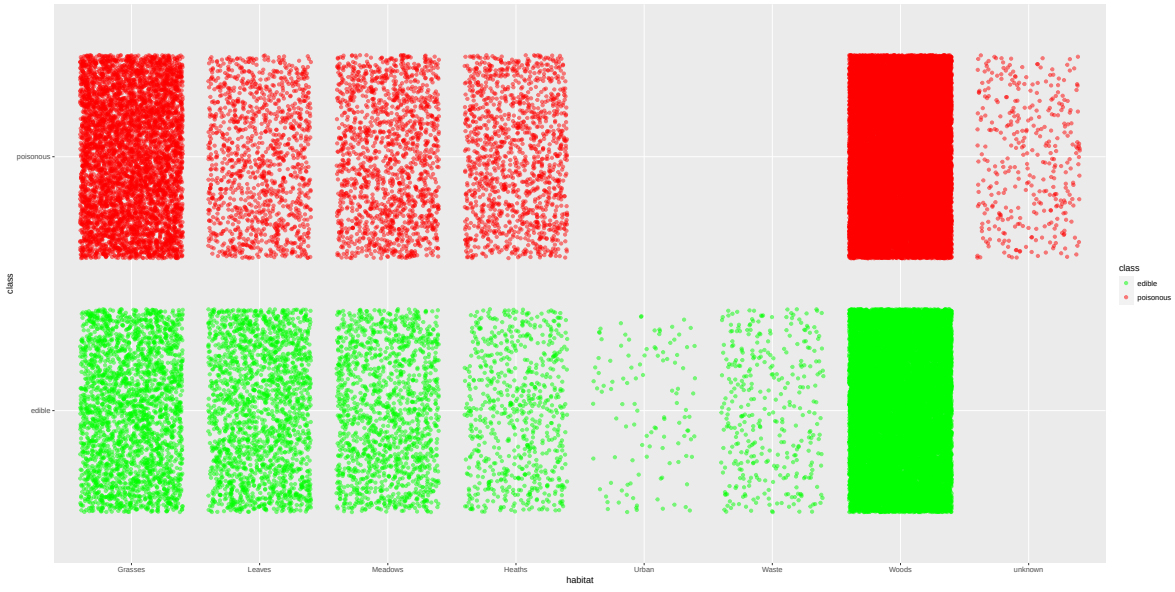


Figure 4: Mushroom classes per habitat

Methodology

The goal of this analysis is to predict if a mushroom is edible or poisonous. In order to achieve this goal, we will compare the performance of different statistical methods in terms of training and testing error rates. Since the dataset contains several categorical variables, one hot encoding is used to encode the categorical features to numeric values. The methods increased the number of columns in the data to 96 columns. We also convert the response variable to 0 for poisonous and 1 for edible. Feature selection was attempted using elastic net, however none of the variables did go to zero. So, I decided to not perform feature selection.

After pre-processing, the data was split into 80% for training and 20% for testing (roughly **48856** for training and **12213** for testing). The main machine learning methods we plan to use in the analysis are Random Forest (rf) and Gradient boosted method (gbm). To compare their performance, we will look at the performance of three baseline models: Naive Bayes, Logistic regression and Support Vector machines.

The first phase of this analysis is to fit the following models:

- Naive Bayes: This probabilistic classifier assumes independence of the predictors and uses the Bayes rule to compute the conditional posterior probabilities of our response variable.
- Logistic Regression: is commonly used for binary classification problems, as it makes no assumptions regarding the distribution of the target classes. However, it can become unstable when there is a clear separation between classes. To predict which class a data point belongs to, a probability cutoff of .5 was initially used since the data is somewhat balanced (48% edible, 52% poisonous). However, after conducting cross-validation using different cutoff probabilities (check code in appendix), a cutoff of **0.45** was used as it achieved the lowest testing error on testing data. So, if the prediction was greater or equal to .45, we assign it to class edible, otherwise we assign it to class poisonous.
- Support Vector Machines: The objective of the support vector machine algorithm is to find a hyper-plane that distinctly classifies the data points. For the two classes case, The objective is to find a plane that has the maximum margin between the two classes. **ksvm()** was used with **rbfdot** kernel and a tuning parameter $\mathbf{C = 100}$. The model was fitted on a scaled data set
- Random Forest: can be used both for regression and classification problems. In this case, this ensemble classifier operates by constructing 250 decision trees and taking the majority vote of these

trees for prediction. As we know,ensemble learners generally outperform all their constituents, however, data characteristics can still affect their performance.

- Gradient Boosting Model: is another ensemble learner that is used for both classification and regression problems. The main concept of this method is to improve the week learners sequentially and increase the model accuracy with a combined model. There are several boosting algorithms such as Gradient boosting, AdaBoost, XGBoost and others. The optimal number of boosting iterations is estimated through cross-validations.

The next phase is to conduct hyper-parameter tuning for both Random Forest and Gradient boosted method. Tuning each algorithm's hyper-parameters is a crucial step of any modeling task. For Random Forest, one can fine-tune parameters such as:

- **ntree** = number of tress to grow, and the default is 500.
- **mtry** = number of variables randomly sampled as candidates at each split. The default is $\sqrt{(p)}$ for classification and $\frac{p}{3}$ for regression.
- **nodesize** = minimum size of terminal nodes. The default value is 1 for classification and 5 for regression
- **sampsize** = Size(s) of sample to draw. For classification, if sampsize is a vector of the length the number of strata, then sampling is stratified by strata, and the elements of sampsize indicate the numbers to be drawn from the strata.

For our analysis, we will only fine tune **ntree** using a **c(50, 100, 150, 200, 250, 300, 350, 400, 500)**, **mtry** using a list from 2 to 5, nodesize using **seq(from=3,to=9,by=2)**, and sampsize of **c(.55, .632, .70, .80)**. The choices were limited because we are dealing with a very large dataset and we are not using any cloud computing.

For the Gradient Boosting Model, we will run a 10-fold cross validation to estimate the optimal number of iterations.

Results

A summary of the initial testing and training errors (no tuning) can be found below:

Table 1: Initial Summary of Training and Testing Errors

	NB	logit	kernel svm	rf	gbm
TrainingErrors	0.47339	0.21424	0.00348	0.00000	0.26492
TestingErrors	0.47196	0.21796	0.00540	0.00041	0.26889

The results of each model's performance is summarized below in the following confusion matrices. The confusion matrices show how many records were misclassified by each model

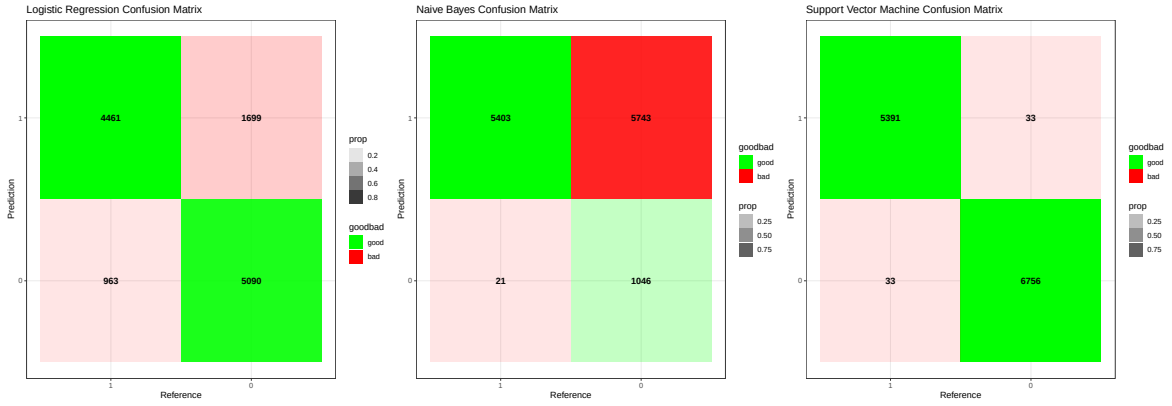


Figure 5: Confusion Matrices for baseline models

We can clearly see that Random Forest is superior to the other models even without tuning. Naive Bayes is the worst performing model with almost 50% testing/training errors. followed by GBM and then logistic regression. Kernel support vector machines has performed the best out of all base models.

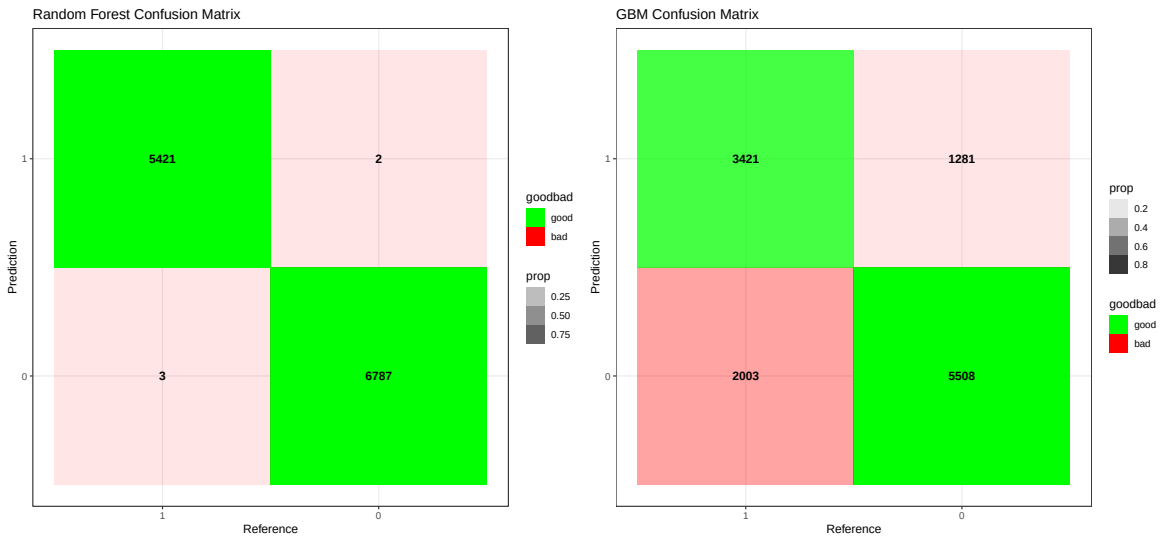


Figure 6: Confusion Matrices

Hyper-parameter tuning

We will start the hyper-parameter tuning by tuning Random forest parameters. We begin tuning the number of trees in the forest then we will move to the other parameters. The table below displays the results of using different number of trees in the random forest model.

Table 2: RF Errors

Trees	Training.Error	Testing.Error
50	0	0.000491
100	0	0.000409
150	0	0.000409
200	0	0.000409
250	0	0.000409
300	0	0.000328
350	0	0.000491
400	0	0.000409
500	0	0.000409

Based on the table above, we see that we achieved the lowest testing/training errors using: 300 trees.

Next, we will tune **mtry**, **nodesize** and **samplesize** using a faster implementation of Random Forest using the R package **"ranger"**

Table 3: RF Cross Validation results

mtry	min.node.size	sample.size	OOB
6	5	0.632	0.123403
6	3	0.632	0.123659
6	7	0.632	0.123722
6	9	0.632	0.123986
6	9	0.800	0.124203
6	3	0.800	0.124305
6	7	0.800	0.124344
6	7	0.700	0.124711
6	5	0.700	0.125097
6	3	0.700	0.125127
6	5	0.800	0.125215
6	9	0.700	0.125431
6	3	0.550	0.125588
6	7	0.550	0.125838
6	9	0.550	0.126027

For GBM, we determine the estimated optimal number of iterations by running a 10-fold cross validation using 4000 trees.

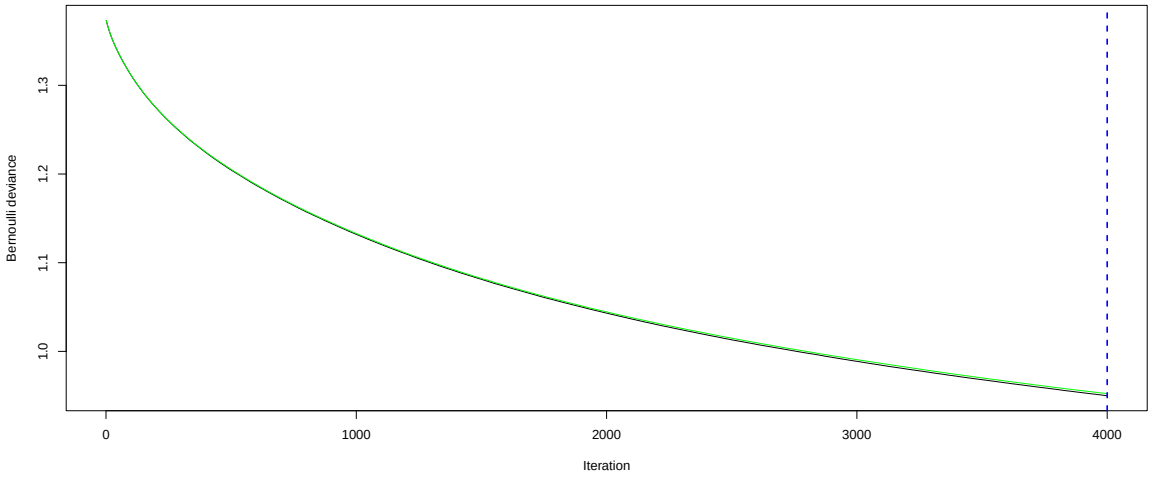


Figure 7: Bernoulli Deviance per Iteration

GBM minimum testing error after tuning is: 0.2103496 and that was achieved using 4000 iterations.

Conclusion and Findings

In this homework, we explored different classification methods. Two out of three baseline models performed poorly on the dataset chosen. It could entirely be because it was roughly composed of categorical variables (90%) and how sparse it's become after one-hot encoding. Additionally, tuning every hyper-parameter for both Random Forest and Gradient Boosting model was impossible given the computing limitations. Random Forest was superior with and without tuning.

Appendix

R Code

```
# Settings
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE, fig.pos = 'H')


# Check if packages are installed. If not, install them.
install.packages <- function(pkg) {
    new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
    if (length(new.pkg)) {
        install.packages(new.pkg)
    }
}


packages <- c("ggplot2", "mcmcplots", "knitr","xtable", "kableExtra","klaR","kernlab","kknn", "missMD
              "mice","inspectdf","broom","ranger", "reshape2", "devtools","MASS","glmnet", "lares",
              "caret","GGally","naniar","VIM","missForest", "FactoMineR",
              "splines", "class", "dplyr", "PerformanceAnalytics",  "factoextra",
              "Amelia", "corrplot", "randomForest")


install.packages(packages)
library(knitr)
library(gbm)
library(kableExtra)
library(ggplot2)
library(devtools)
library(extrafont)
library(dplyr)
library(tidyverse)
library(VIM)
library(randomForest)
library(naivebayes)
library(caret)
```

```r
library(MASS)
library(klaR)
library(kernlab)
library(class)
library(missForest)
library(FactoMineR)
library(inspectdf)
library(GGally)
library(patchwork)
library(tidyr)
library(factoextra)
options(kableExtra.latex.load_packages = FALSE)
loadfonts()
# read data
mushroom <- read.table("MushroomDataset/secondary_data.csv", head=T, sep=";")

# mushroom <- read.csv("sample_data.csv")

# convert stem height from cm to mm
mushroom$stem.height <- 10 * mushroom$stem.height

# convert cap diameter from cm to mm
mushroom$cap.diameter <- 10 * mushroom$cap.diameter

# converts all categorical data to factors
mushroom <- mushroom %>% mutate_if(is.character,as.factor)

# rename the levels for all the categorical columns for plotting
levels(mushroom$class) <- c("edible", "poisonous")

levels(mushroom$habitat) <-  list(Grasses  = "g", Leaves = "l", Meadows = "m", Heaths = "h",
                                  Urban = "u", Waste = "w", Woods = "d")

levels(mushroom$season) <-  list(Spring  = "s", Summer = "u", Autumn = "a", Winter = "w")
```

```
levels(mushroom$cap.shape)  <- list(bell= "b", conical="c", convex="x", flat="f",sunken="s", spherica

levels(mushroom$cap.surface)  <- list(fibrous="i", grooves="g", scaly="y", smooth="s", shiny="h",
                                      leathery="l", silky="k",sticky="t", wrinkled="w", fleshy="e")

levels(mushroom$cap.color) <- list(brown="n", buff="b", gray="g", green="r", pink="p",purple="u",
                                   red="e", white="w", yellow="y", blue="l",orange="o",  black="k")

levels(mushroom$spore.print.color) <- list(brown="n", buff="b", gray="g", green="r", pink="p",purple=
                                   red="e", white="w", yellow="y", blue="l",orange="o",  black="k")
levels(mushroom$stem.color) <-  list(brown="n", buff="b", gray="g", green="r", pink="p",purple="u",
                                   red="e", white="w", yellow="y", blue="l",orange="o",  black="k", n

levels(mushroom$stem.surface)  <- list(fibrous="i", grooves="g", scaly="y", smooth="s", shiny="h",
                                      leathery="l", silky="k",sticky="t", wrinkled="w", fleshy="e", no

levels(mushroom$veil.color) <-  list(brown="n", buff="b", gray="g", green="r", pink="p",purple="u",
                                   red="e", white="w", yellow="y", blue="l",orange="o",  black="k", n

levels(mushroom$veil.type) <-  list(partial="p", universal="u")

levels(mushroom$has.ring) <- list(ring="t", none="f")
levels(mushroom$ring.type) <- list(cobwebby='c', evanescent='e', flaring='r', grooved='g',large='l',

levels(mushroom$gill.spacing) <- list(close="c", distant="d", none="f")

levels(mushroom$gill.color) <-  list(brown="n", buff="b", gray="g", green="r", pink="p",purple="u",
                                   red="e", white="w", yellow="y", blue="l",orange="o",  black="k", n

levels(mushroom$does.bruise.or.bleed) <- list(bruises.or.bleeding='t', no='f')

levels(mushroom$gill.attachment) <- list(adnate='a', adnexed='x', decurrent='d', free='e', sinuate='s
```

```
levels(mushroom$stem.root) <- list(bulbous='b', swollen='s', club='c', cup='u', equal='e',rhizomorphs

# check for missing values
aggr(mushroom, col = c('green', 'red'), numbers = T, sortVars = T)
# drop columns with more than 70% missing values
mushroom.clean <- mushroom[, which(colMeans(!is.na(mushroom))> 0.65)]

# replace missing values in categorical vars by "unknown
mushroom.clean <- mushroom.clean %>% mutate_if(is.factor, ~ fct_explicit_na(., na_level = "unknown"))

# show missing values
agg_missing_values <- aggr(mushroom.clean, col = c('green', 'red'), numbers = T, sortVars = T)


temp <- mushroom.clean[, c('class', 'cap.diameter', 'stem.height', 'stem.width')]

ggpairs(temp,legend = 1,
               mapping = ggplot2::aes(colour=class)) +
               theme(legend.position = "bottom")


ggplot(mushroom.clean, aes(x = cap.surface, y = cap.color, col = class)) +
  geom_jitter(alpha = 0.6) +
  scale_color_manual(breaks = c("edible", "poisonous"),
                     values = c("green", "red"))

ggplot(mushroom.clean, aes(x = cap.color, y = cap.shape, col = class)) +
  geom_jitter(alpha = 0.6) +
  scale_color_manual(breaks = c("edible", "poisonous"),
                     values = c("green", "red"))

ggplot(mushroom.clean, aes(x = habitat, y = class , col = class)) +
  geom_jitter(alpha = 0.5) +
  scale_color_manual(breaks = c("edible", "poisonous"),
```

```
                    values = c("green", "red"))
library(tibble)
dummy_mush <- dummyVars(~ ., data = mushroom.clean[, -1])
mushroom_updated <- as_tibble(predict(dummy_mush, newdata = mushroom.clean))
mushroom_updated <- cbind(target = as.factor(mushroom.clean$class), mushroom_updated)

# rename class column to avoid confusion with the class library
mushroom.official <- mushroom_updated
mushroom.official$target <- ifelse(mushroom.official$target == "edible", 1, 0)
set.seed(42)
ind <- caret::createDataPartition(y = mushroom.official$target, times = 1, p = 0.8, list = FALSE)
mushTrain <- mushroom.official[ind, ]
mushTest <- mushroom.official[-ind, ]

training <- mushTrain[,-1]

testing <- mushTest[,-1]

ytrain <- mushTrain$target
ytest <- mushTest$target

TestingErrors  <- NULL;
TrainingErrors <- NULL;

##### Fit baseline models ##############

## Naive Bayes
bayes.model <- naive_bayes(as.factor(target) ~ ., data=mushTrain,  usekernel = T)

pred1test <- predict(bayes.model,newdata=testing)
pred1train <- predict(bayes.model,newdata=training)

mod1train <- mean(pred1train != ytrain)
mod1test <- mean(pred1test != ytest)
```

```
TestingErrors <- c(TestingErrors, mod1test);
TrainingErrors <- c(TrainingErrors, mod1train);

# Confusion Matrix
bayes.table <- data.frame(confusionMatrix(as.factor(pred1test) ,as.factor(ytest),
                                    dnn = c("Prediction", "Reference"))$table)

bayes.plotTable <- bayes.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

bayes.cm <- ggplot(data = bayes.plotTable, mapping = aes(x = Reference,
                                        y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("Naive Bayes Confusion Matrix") +
  xlim(rev(levels(bayes.plotTable$Reference)))


### Logistic regression
glm.model <- glm(as.numeric(target) ~ ., data=mushTrain, family = binomial(link="logit"));

pred2test <- predict(glm.model, newdata=testing,type="response")
pred2test <- ifelse(pred2test > 0.45, 1, 0)

pred2train <- predict(glm.model, newdata=training,type="response")
pred2train <- ifelse(pred2train > 0.45, 1,0)

mod2train <- mean(pred2train != ytrain)
mod2test <- mean(pred2test != ytest)
```

```
TestingErrors <- c(TestingErrors, mod2test);
TrainingErrors <- c(TrainingErrors, mod2train);

# Confusion Matrix
glm.table <- data.frame(confusionMatrix(as.factor(pred2test) ,as.factor(ytest),
                                        dnn = c("Prediction", "Reference"))$table)

glm.plotTable <- glm.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

glm.cm <- ggplot(data = glm.plotTable, mapping = aes(x = Reference,
                                           y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("Logistic Regression Confusion Matrix") +
  xlim(rev(levels(glm.plotTable$Reference)))

#### SVM
svm.model <- ksvm(as.factor(target) ~., data=mushTrain,type="C-svc",kernel="rbfdot",C=100,scaled=T)

pred4test <- predict(svm.model,newdata=testing)
mod4test <- mean(pred4test != ytest)

pred4train <- predict(svm.model,newdata=training)
mod4train <- mean(pred4train != ytrain)

TestingErrors <- c(TestingErrors, mod4test);
TrainingErrors <- c(TrainingErrors, mod4train);
```

```r
# Confusion Matrix
svm.table <- data.frame(confusionMatrix(as.factor(pred4test) ,as.factor(ytest),
                                        dnn = c("Prediction", "Reference"))$table)


svm.plotTable <- svm.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))


svm.cm <- ggplot(data = svm.plotTable, mapping = aes(x = Reference,
                                           y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("Support Vector Machine Confusion Matrix") +
  xlim(rev(levels(svm.plotTable$Reference)))


### Random Forest
rf.model <- randomForest(as.factor(target) ~ ., ntree = 200, mtry=32, data = mushTrain, importance=TR


pred3test <- predict(rf.model,newdata=testing, type="class")
mod3test <- mean(pred3test != ytest)


pred3train <- predict(rf.model,newdata=training, type="class")
mod3train <- mean(pred3train != ytrain)


TestingErrors <- c(TestingErrors, mod3test);
TrainingErrors <- c(TrainingErrors, mod3train);


# Confusion Matrix
rf.table <- data.frame(confusionMatrix(as.factor(pred3test) ,as.factor(ytest),
                                        dnn = c("Prediction", "Reference"))$table)
```

```
rf.plotTable <- rf.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

rf.cm <- ggplot(data = rf.plotTable, mapping = aes(x = Reference,
                                        y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("Random Forest Confusion Matrix") +
  xlim(rev(levels(rf.plotTable$Reference)))

### GBM
gbm.model <- gbm(as.character(target) ~ .,data = mushTrain,
                 distribution = 'bernoulli')

# Make Prediction
pred6test <- predict(gbm.model, newdata=testing, type="response")
mod6test <- mean(round(pred6test, 0) != ytest)

pred6train <- predict(gbm.model, newdata=training, type="response")
mod6train <- mean(round(pred6train, 0) != ytrain)

TestingErrors <- c(TestingErrors, mod6test);
TrainingErrors <- c(TrainingErrors, mod6train);

# Confusion Matrix
gbm.table <- data.frame(confusionMatrix(as.factor(round(pred6test, 0)) ,as.factor(ytest),
                                        dnn = c("Prediction", "Reference"))$table)

gbm.plotTable <- gbm.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
```

```
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))


gbm.cm <- ggplot(data = gbm.plotTable, mapping = aes(x = Reference,
                                        y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("GBM Confusion Matrix") +
  xlim(rev(levels(gbm.plotTable$Reference)))


error_summary <- rbind(TrainingErrors, TestingErrors)
colnames(error_summary) <- c("NB","logit", "kernel svm", "rf", "gbm")


# displaying the tables
knitr::kable(error_summary,"pipe", digit=5, format.args = list(scientific = FALSE),
             caption = paste("Initial Summary of Training and Testing Errors"))%>%
        kable_styling(position = "center", full_width = F)


glm.cm  + bayes.cm + svm.cm
rf.cm + gbm.cm


set.seed(420)
RFtrainError <- c()
RFtestError <- c()
trees <- c(50, 100, 150, 200, 250, 300, 350, 400, 500)
for (ntree in trees) {


  fit <- randomForest(as.factor(target) ~ ., ntree = ntree, mtry=32,data = mushTrain, importance=TRUE
  predTest <- predict(fit,newdata=testing, type="class")
  predTrain <- predict(fit,newdata=training, type="class")


  # training and testing errors
```

```
  modtrain <- mean(predTrain != ytrain)
  modtest <- mean(predTest != ytest)
    RFtrainError <- c(RFtrainError, modtrain)
    RFtestError <- c(RFtestError, modtest)

}
# plot
test_error_tbl <- data.frame(
  Trees = trees,
  Testing.Error = RFtestError
)

train_error_tbl <- data.frame(
  Trees = trees,
  Training.Error = RFtrainError
)



error_data = cbind(train_error_tbl,test_error_tbl)
knitr::kable(error_data[, c(1, 2, 4)],"pipe", digit=6,format.args = list(scientific = FALSE),
             caption = "RF Errors")%>%
        kable_styling(position = "center", full_width = F)

library(ranger)       # a faster implementation of random Forest

grid <- expand.grid(mtry=2:6,min.node.size=seq(3,9,2),sample.size=c(.55, .632, .70, .80))

for (i in 1:nrow(grid)) {

  model <- ranger(target ~ .,
                  data=mushTrain,
                  num.trees = 500,
                  mtry=grid$mtry[i],
                  min.node.size = grid$min.node.size[i],
```

```
                  sample.fraction =grid$sample.size[i],
                  seed=123,
                  write.forest = T)

  grid$OOB[i] <- sqrt(model$prediction.error)
}

#
top15 <- grid %>% dplyr::arrange(OOB)%>%head(15)

knitr::kable(top15,"pipe", digit=6, format.args = list(scientific = FALSE),
             caption = "RF Cross Validation results ")%>%
        kable_styling(position = "center", full_width = F)

set.seed(1)

### GBM
gbm.model <- gbm(as.character(target) ~ .,data = mushTrain,
                distribution = 'bernoulli',
                   n.trees = 4000, shrinkage = 0.01, cv.folds = 10)


## Find the estimated optimal number of iterations
perf_gbm <- gbm.perf(gbm.model, method="cv")

## summary model

# ## Make Prediction
pred <- predict(gbm.model, newdata=testing, n.trees=perf_gbm ,type="response")
gbmtest <- mean(round(pred, 0) != ytest)

#### Find the best prob cutoff that reduces the testing error

# THRESHOLD <- seq(0.1, 0.95, by = 0.05)
```

```r
# accuracy_list <- c()
# precision_list <- c()
# recall_list <- c()
# testingError_list <- c()
# glm.model <- glm(as.factor(target) ~ ., data=mushTrain, family = binomial(link="logit"));
# for (i in THRESHOLD) {
#
#   prediction <- as.integer(predict(glm.model, newdata=testing,type="response") > i)
#   reference <- as.factor(ytest)
#   confusion_table <- table(as.factor(prediction), reference)
#   n <- sum(confusion_table) # number of instances
#   diag <- diag(confusion_table)
#   accuracy <- sum(diag) / n # Calculate the Accuracy
#   TP <- confusion_table[2,2]
#   FP <- confusion_table[1,2]
#   FN <- confusion_table[2,1]
#   precision <- TP/(TP+FP) # Calculate the Precision
#   recall <- TP/(TP+FN) # Calculate the Recall
#   error <- mean(prediction != reference)
#   accuracy_list <- c(accuracy_list, accuracy)
#   precision_list <- c(precision_list, precision)
#   recall_list <- c(recall_list, recall)
#   testingError_list <- c(testingError_list, error)
# }
# cat("Maximum accuracy obtained is ", max(accuracy_list) * 100, "% \n")
# cat("Probability threshold with the max accuracy is: ", THRESHOLD[which.max(accuracy_list)]) # 0.45
# cat("Probability threshold with the minimum testing error is: ", THRESHOLD[which.min(testingError_l
```