FIT3152 Data analytics

Assignment 2

Name: Lim Yu-Shan

Student ID: 32685467

Notes to marker:

- The main body of this report is almost 15 pages. This length is due to some long code blocks and the sizes of the decision tree diagram and ROC curve plots that have to be big to make them readable. All other pages are the Appendix, which include repeated code and outputs.
- Due to length, from Question 9 onwards, similar code from Questions 4 and 5 used to make predictions, create confusion matrices, report accuracies, construct ROC curves and compute AUC value will not be shown in the main report but in the Appendix. ROC curve plots will also not be shown in the main report except that of Question 12, which shows the curves for all classifiers.

My unique dataset of 2000 rows is first created with the following code from the specifications. Libraries needed to perform the tasks are imported as well.

```
rm(list = ls())
WAUS <- read.csv("HumidPredict2023D.csv", stringsAsFactors = TRUE)
L <- as.data.frame(c(1:49))
set.seed(32685467)
L <- L[sample(nrow(L), 10, replace = FALSE), ]
WAUS <- WAUS[(WAUS$Location %in% L), ]
WAUS <- WAUS[sample(nrow(WAUS), 2000, replace = FALSE), ]

library(dplyr)
library(caret)
library(tree)
library(e1071)
library(adabag)
library(randomForest)
library(ROCR)
library(lightgbm)
library(kernlab)
```

Question 1

The proportion of days when it is more humid than the previous day, and the opposite, can be computed by subsetting the data based on the value of `MHT`, which indicates whether the day in the next entry is more humid than that of the current one.

```
more <- nrow(subset(WAUS, subset = MHT == 1))
less <- nrow(subset(WAUS, subset = MHT == 0))
more
```

```
## [1] 958
```

```
less
```

```
## [1] 933
```

```
more / less
```

```
## [1] 1.026795
```

```
nrow(WAUS[is.na(WAUS$MHT), ])
```

```
## [1] 109
```

Based on the results, the proportion is about 1.03, which means their respective frequencies are close to equal. The `MHT` field is also empty for 109 entries, so the true proportion may differ slightly as roughly 5% of the data has no target variable value.

`summary()` is then run to get descriptions of the predictors. Due to its length, the output is shown in the Appendix.

```
summary(WAUS)
```

Among the numerical real-valued attributes, most of the means are between 2 and 22. Some exceptions are `WindGustSpeed`, with a mean of 43.62, and more notably, `Pressure9am` and `Pressure3pm`, with means more than 1000 (which is normal for the scale at which atmospheric pressure is measured). Some interesting mean values are `Rainfall`'s mean of 2.941 despite having a max value of 156.4, and `WindGustSpeed`'s mean despite having a max value of 130. In both cases, this indicates a presence of a few extreme values.

Additionally, one noteworthy observation is that there are NA values for every predictor except `Location`, with some predictors having significantly more than the rest (eg. `Sunshine` has 842 NA's compared to `Year`'s 22).

Then, this code is run to compute the standad deviations of the numerical attributes.

```
apply(WAUS, 2, sd, na.rm = TRUE)
```

```
##          Year      Location       MinTemp       MaxTemp      Rainfall
##     3.3238219    11.8120490     7.1964777     8.0313450    10.0762448
##   Evaporation      Sunshine   WindGustDir WindGustSpeed    WindDir9am
##     2.9161313     3.7131284            NA    14.5964117            NA
##    WindDir3pm  WindSpeed9am  WindSpeed3pm   Pressure9am   Pressure3pm
##            NA     8.9975163     8.4656040     7.6211719     7.5892060
##      Cloud9am      Cloud3pm       Temp9am       Temp3pm     RainToday
##     2.6995993     2.6245726     7.4366671     7.9707124            NA
##       RISK_MM           MHT
##     9.2276771     0.5000885
```

As expected, `WindGustSpeed` and `Rainfall` are the two real-valued attributes with the largest standard deviations due to their extreme values. The smallest standard deviation belongs to `Cloud3pm`, meaning that the fraction of sky obscured by cloud at 3pm has been consistent, for this dataset.

The only attributes that may, from first glance, be worth omitting are `Location` and `Year`, as a country's yearly climate is usually expected to be consistent. But due to Australia's size, diverse climate and geography, and climate change in recent years, both `Location` and `Year` might have a significant influence on `MHT` and should be kept. All other predictors are closely related to the weather and are thus assumed to have predictive power on `MHT` as well.

Question 2

As all the predictors seem to have relatively equal importance based on their descriptions in the specifications (and it is generally a good idea to to build initial classifers using all attributes and skim them down from there), no columns are dropped. The only pre-processing required is to remove all rows with NA values to make the data suitable for model fitting.

The following code extracts only complete entries from `WAUS` into a new data frame called `waus_clean`, and `MHT` is also converted to factors as its data type is numerical, which would result in a regression tree being built instead of a classification tree that classifies data points into 0 and 1.

```
waus_clean <- WAUS[complete.cases(WAUS), ]
waus_clean$MHT <- as.factor(waus_clean$MHT)

nrow(waus_clean)
```

```
## [1] 655
```

```
table(waus_clean$Location)
```

```
##
##  14  16  28  38  39
## 165 130 117  92 151
```

The outputs of the last two lines shows that the processed dataset now contains 655 rows, and the number of unique locations has been reduced to 5.

Question 3

The adapted code to divide my data with a 70% to 30% train-test split is as follows.

```
train_row <- sample(1:nrow(waus_clean), 0.7 * nrow(waus_clean))
waus_train <- waus_clean[train_row, ]
waus_test <- waus_clean[-train_row, ]
```

Question 4

A classification model of each technique is constructed with default settings.

```
# decision tree
waus_tree <- tree(MHT ~ ., data = waus_train)

# naive bayes
waus_bayes <- naiveBayes(MHT ~ ., data = waus_train)

# bagging
waus_bag <- bagging(MHT ~ ., data = waus_train)

# boosting
waus_boost <- boosting(MHT ~ ., data = waus_train)

# random forest
waus_forest <- randomForest(MHT ~ ., data = waus_train)
```

Question 5

Each data point in the test data is classified into either `1` (more humid tomorrow) or `0` (less humid tomorrow) using `predict()` (`predict.bagging()` and `predict.boosting()` for bagging and boosting respectively).

```
# decision tree
waus_tree_predict <- predict(waus_tree, waus_test, type = "class")

# naive bayes
waus_bayes_predict <- predict(waus_bayes, waus_test)

# bagging
waus_bag_predict <- predict.bagging(waus_bag, waus_test)
```

```
# boosting
waus_boost_predict <- predict.boosting(waus_boost, waus_test)

# random forest
waus_forest_predict <- predict(waus_forest, waus_test)
```

A confusion matrix is then created for each model, displayed alongside each model's accuracy, which is the total number of accurate predictions divided by total number of observations. A function is used to compute the accuracy for each confusion matrix, using the code `sum(diag(cm)) / sum(cm)`, where `cm` is a given confusion matrix.

```
get_accuracy <- function(cm) {
    return(sum(diag(cm)) / sum(cm))
}

# decision tree
waus_tree_cm <- table("Predicted Class" = waus_tree_predict,
                      "Actual Class" = waus_test$MHT)
waus_tree_acc <- get_accuracy(waus_tree_cm)
cat("Decision tree (accuracy:", waus_tree_acc, ")\n")
```

```
## Decision tree (accuracy: 0.5532995 )
```

```
waus_tree_cm
```

```
##                Actual Class
## Predicted Class  0  1
##               0 40 39
##               1 49 69
```

```
# naive bayes
waus_bayes_cm <- table("Predicted Class" = waus_bayes_predict,
                       "Actual Class" = waus_test$MHT)
waus_bayes_acc <- get_accuracy(waus_bayes_cm)
cat("Naive bayes classifier (accuracy:", waus_bayes_acc, ")\n")
```

```
## Naive bayes classifier (accuracy: 0.5482234 )
```

```
waus_bayes_cm
```

```
##                Actual Class
## Predicted Class  0  1
##               0 28 28
##               1 61 80
```

```
# bagging
waus_bag_acc <- get_accuracy(waus_bag_predict$confusion)
cat("Bagging (accuracy:", waus_bag_acc, ")\n")
```

```
## Bagging (accuracy: 0.6345178 )
```

```
waus_bag_predict$confusion
```

```
##                Observed Class
## Predicted Class  0  1
##               0 47 30
##               1 42 78
```

```r
# boosting
waus_boost_acc <- get_accuracy(waus_boost_predict$confusion)
cat("Boosting (accuracy:", waus_boost_acc, ")\n")
```

```
## Boosting (accuracy: 0.6345178 )
```

```r
waus_boost_predict$confusion
```

```
##                 Observed Class
## Predicted Class  0  1
##               0 52 35
##               1 37 73
```

```r
# random forest
waus_forest_cm <- table("Predicted Class" = waus_forest_predict,
                        "Actual Class" = waus_test$MHT)
waus_forest_acc <- get_accuracy(waus_forest_cm)
cat("Random forest (accuracy:", waus_forest_acc, ")\n")
```

```
## Random forest (accuracy: 0.5939086 )
```

```r
waus_forest_cm
```

```
##                 Actual Class
## Predicted Class  0  1
##               0 27 18
##               1 62 90
```

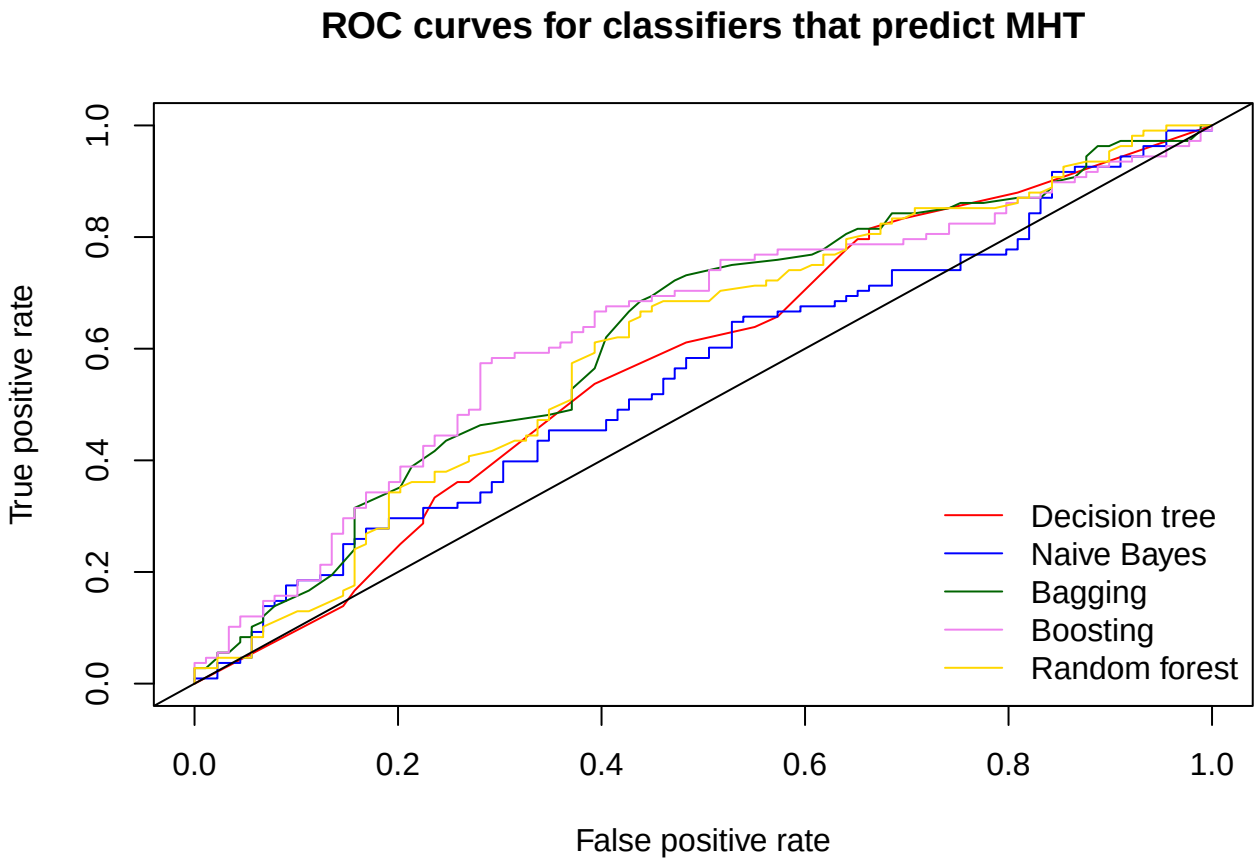
Among the classifiers, only bagging and boosting have an accuracy above 0.6. In fact, they have identical accuracies (0.6345178). The other three classifiers have accuracies closer to that of random guessing, with the naive Bayes classifier having the worst accuracy at 0.5482234.

Question 6

The confidence of predicting 'more humid tomorrow' is computed by adjusting the `type` parameter of the `predict()` function (`predict.bagging()` and `predict.boosting()` for bagging and boosting respectively) for each model. Their respective ROC curves is then computed using `prediction()` (which standardises the input data) and `performance()` (which evaluates a model's true positive rate and false positive rate). The ROC curves are then plotted on the same axis with different colours, and a diagonal line is added to represent the random classifier (for comparison purposes).

```r
# decision tree
waus_tree_predict_prob <- predict(waus_tree, waus_test, type = "vector")
waus_tree_pred <- prediction(waus_tree_predict_prob[, 2], waus_test$MHT)
waus_tree_perf <- performance(waus_tree_pred, "tpr", "fpr")
plot(waus_tree_perf, col = "red")

# naive bayes
waus_bayes_predict_prob <- predict(waus_bayes, waus_test, type = "raw")
waus_bayes_pred <- prediction(waus_bayes_predict_prob[, 2], waus_test$MHT)
waus_bayes_perf <- performance(waus_bayes_pred, "tpr", "fpr")
plot(waus_bayes_perf, col = "blue", add = TRUE)

# bagging
waus_bag_predict_prob <- predict.bagging(waus_bag, waus_test, type = "prob")
waus_bag_pred <- prediction(waus_bag_predict_prob$prob[, 2], waus_test$MHT)
waus_bag_perf <- performance(waus_bag_pred, "tpr", "fpr")
```

```
plot(waus_bag_perf, col = "darkgreen", add = TRUE)

# boosting
waus_boost_predict_prob <- predict.boosting(waus_boost, waus_test, type = "prob")
waus_boost_pred <- prediction(waus_boost_predict_prob$prob[, 2], waus_test$MHT)
waus_boost_perf <- performance(waus_boost_pred, "tpr", "fpr")
plot(waus_boost_perf, col = "violet", add = TRUE)

# random forest
waus_forest_predict_prob <- predict(waus_forest, waus_test, type = "prob")
waus_forest_pred <- prediction(waus_forest_predict_prob[, 2], waus_test$MHT)
waus_forest_perf <- performance(waus_forest_pred, "tpr", "fpr")
plot(waus_forest_perf, col = "gold", add = TRUE)

abline(0, 1)
legend("bottomright",
       c("Decision tree", "Naive Bayes", "Bagging", "Boosting", "Random forest"),
       col = c("red", "blue", "darkgreen", "violet", "gold"),
       lty = 1, bty = "n", inset = c(0, 0))
title("ROC curves for classifiers that predict MHT")
```



The AUC for each classifier is also calculated using `performance()`, but with the parameters adjusted to obtain the `auc` metric.

```
waus_tree_auc <- performance(waus_tree_pred, "auc")@y.values[[1]]
waus_bayes_auc <- performance(waus_bayes_pred, "auc")@y.values[[1]]
waus_bag_auc <- performance(waus_bag_pred, "auc")@y.values[[1]]
```

```
waus_boost_auc <- performance(waus_boost_pred, "auc")@y.values[[1]]
waus_forest_auc <- performance(waus_forest_pred, "auc")@y.values[[1]]

cat("Decision tree AUC", waus_tree_auc, "\n")
```

```
## Decision tree AUC 0.5761548
```

```
cat("Naive Bayes classifier AUC", waus_bayes_auc, "\n")
```

```
## Naive Bayes classifier AUC 0.5527466
```

```
cat("Bagging AUC", waus_bag_auc, "\n")
```

```
## Bagging AUC 0.6255202
```

```
cat("Boosting tree AUC", waus_boost_auc, "\n")
```

```
## Boosting tree AUC 0.6351436
```

```
cat("Random forest AUC", waus_forest_auc, "\n")
```

```
## Random forest AUC 0.6069496
```

If the classifiers are arranged in ascending order of AUC value, they would be in ascending order of accuracy as well, except that bagging and boosting have different AUC values. Boosting has an AUC value of 0.6351436, a little higher than that of bagging (0.6255202).

Question 7

A table to compare the results in Questions 5 and 6, namely the accuracies and AUC values of each classifier, is created in the form of a data frame, using the code as follows.

```
waus_acc_auc <- data.frame(
  model = c("Decision tree", "Naive Bayes classifier", "Bagging", "Boosting",
            "Random forest"),
  accuracy = c(waus_tree_acc, waus_bayes_acc, waus_bag_acc, waus_boost_acc,
               waus_forest_acc),
  auc = c(waus_tree_auc, waus_bayes_auc, waus_bag_auc, waus_boost_auc, waus_forest_auc)
)

waus_acc_auc
```

```
##                    model  accuracy       auc
## 1          Decision tree 0.5532995 0.5761548
## 2 Naive Bayes classifier 0.5482234 0.5527466
## 3                Bagging 0.6345178 0.6255202
## 4               Boosting 0.6345178 0.6351436
## 5          Random forest 0.5939086 0.6069496
```

From the output, there seems to exist a single "best" classifier based on the two metrics in focus, and that would be **boosting**. Both bagging and boosting have identical accuracies, but boosting has a slight edge due to its higher AUC value. However, it cannot be assumed now that boosting is definitely the single best classifier, due to the possibility of overfitting, which is a result of fitting too many attributes into a model.

Question 8

By inspecting the decision tree as follows,

```
summary(waus_tree)
```

```
##
## Classification tree:
## tree(formula = MHT ~ ., data = waus_train)
## Variables actually used in tree construction:
## [1] "Sunshine"     "WindDir9am"   "WindGustDir"  "Rainfall"     "WindDir3pm"
## [6] "Cloud9am"     "Pressure9am"  "MinTemp"      "WindSpeed9am"
## Number of terminal nodes:  22
## Residual mean deviance:  0.8791 = 383.3 / 436
## Misclassification error rate: 0.2511 = 115 / 458
```

it is known that this classifier identifies the following attributes as the most important in predicting whether the next day would be more humid or not: `Sunshine`, `WindDir9am`, `WindGustDir`, `Rainfall`, `WindDir3pm`, `Cloud9am`, `Pressure9am`, `MinTemp` and `WindSpeed9am`.

As a naive Bayes classifier assumes independence between predictors, the predictors have no measure of importance.

For bagging, variable importance, in ascending order, can be displayed with the following code.

```
sort(waus_bag$importance)
```

```
##     RainToday      Location          Year       Temp9am       RISK_MM
##     0.0000000     0.3500016     0.8399457     1.2742170     1.6164396
##      Cloud9am       Temp3pm  WindSpeed3pm WindGustSpeed       MaxTemp
##     1.8075635     1.8854723     1.9166833     1.9741943     2.2822101
##   Pressure9am       MinTemp      Cloud3pm   Evaporation  WindSpeed9am
##     2.3215575     2.3244029     2.4031687     2.4077919     2.5969663
##   Pressure3pm      Rainfall      Sunshine   WindGustDir    WindDir3pm
##     2.6086759     4.2857422     6.7237428    18.8087757    20.0269636
##    WindDir9am
##    21.5454851
```

The output shows that `WindGustDir`, `WindDir3pm` and `WindDir9am` are significantly more important compared to the other attributes. The least important predictor among these three, `WindGustDir` (with a weightage of 18.8087757), alone is nearly three times more important compared to the next less important predictor, `Sunshine` (with a weightage of 6.7237428).

The process is repeated for boosting.

```
sort(waus_boost$importance)
```

```
##     RainToday      Location      Cloud9am      Cloud3pm      Rainfall
##     0.0000000     0.8328512     1.6946388     2.2491230     2.4030670
##  WindSpeed3pm       RISK_MM       MinTemp       Temp9am          Year
##     2.4317427     2.4785325     2.6360581     2.7468566     2.7497278
##       Temp3pm WindGustSpeed   Pressure3pm   Pressure9am  WindSpeed9am
##     2.7762501     2.8456079     3.1109357     3.8239209     3.8559576
##   Evaporation       MaxTemp      Sunshine   WindGustDir    WindDir3pm
##     3.9524535     4.2742923     5.3837980    15.8714548    16.4166275
##    WindDir9am
##    17.4661038
```

Similarly, the boosting classifier considers `WindGustDir`, `WindDir3pm` and `WindDir9am` as significantly more important than the other attributes. In a similar fashion to bagging, `WindGustDir` is the least important among these three and `Sunshine` is the next less important predictor. However, `Sunshine`'s weightage (5.3837980) is also nearly three times less than that of `WindGustDir` (15.8714548).

Finally, the variable importance for the random forest is inspected as well.

```
waus_forest$importance
```

```
##                MeanDecreaseGini
## Year                   7.209436
## Location               3.162052
## MinTemp               10.149433
## MaxTemp               10.538502
## Rainfall               7.594635
## Evaporation           10.449162
## Sunshine              13.295363
## WindGustDir           25.534251
## WindGustSpeed          8.968121
## WindDir9am            27.727453
## WindDir3pm            24.537091
## WindSpeed9am           8.541093
## WindSpeed3pm           7.884748
## Pressure9am           10.632436
## Pressure3pm           11.400200
## Cloud9am               6.459139
## Cloud3pm               6.875054
## Temp9am                9.480876
## Temp3pm               10.279417
## RainToday              1.773276
## RISK_MM                5.015282
```

Again, `WindGustDir`, `WindDir3pm` and `WindDir9am` are the most important, being the only predictors whose weightages are more than 20.

In conclusion, based on these classifiers, `WindGustDir`, `WindDir3pm` and `WindDir9am` are the most important variables in predicting MHT. The decision tree considers other attributes important as well, which are `Sunshine`, `Rainfall`, `Cloud9am`, `Pressure9am`, `MinTemp` and `WindSpeed9am`. However, the importance of these attributes may be exaggerated as the decision tree does not possess good performance metrics (both accuracy and AUC value are less than 0.6), and decision trees that have been fitted using all attributes tend to overfit. That being said, all other variables yet to be mentioned can be omitted from the data with very little effect on performance, as their values provide little to no impact on the outcome of the classification.

Question 9

To create a simple classifier for a person to make classifications by hand, I started from the decision tree created in Question 4. The approach here is to prune the tree to an optimal size, yielding a simpler decision tree that can be used to make classifications easily.

To determine what size to prune the tree to, cross validation is carried out on the original decision tree using `cv.tree()`. All attributes from the original tree is preserved, as the tree was trained on all attributes and, despite possibly overfitted, is the most "complete" (contains the most information) tree to prune. In other words, **post-pruning** will be performed on the original tree to obtain the simple tree.

```
waus_tree_cv <- cv.tree(waus_tree)
waus_tree_cv
```

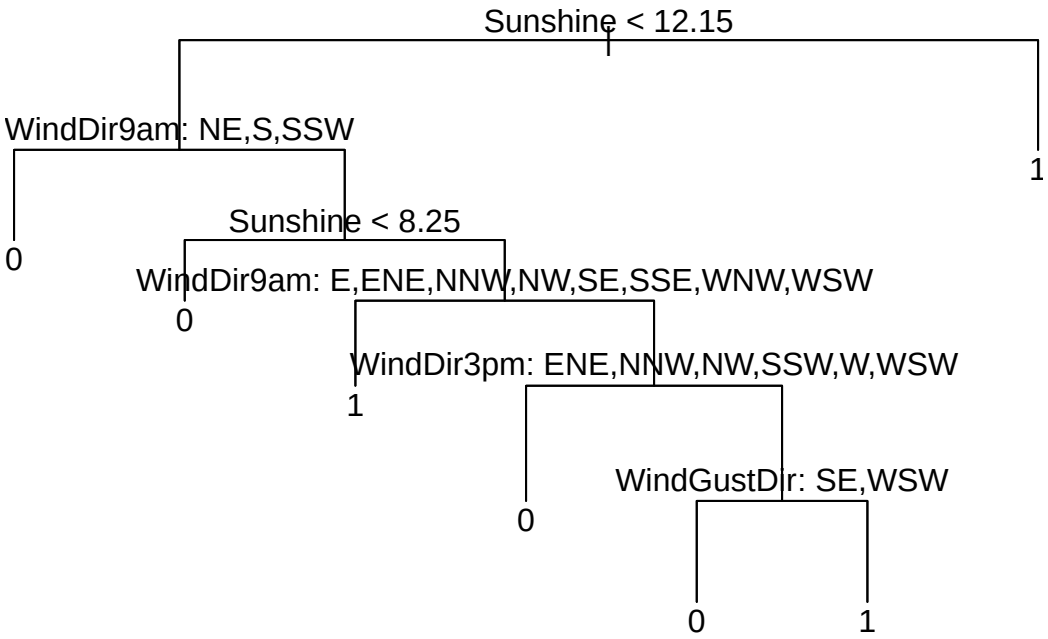
```
## $size
##  [1] 22 21 19 17 16 15 12 10  7  2  1
##
## $dev
##  [1] 1723.8731 1508.1828 1416.5571 1284.0608 1275.0283 1004.6979  965.7242
```

```
##  [8]  800.3798  798.9960  754.2189  685.0633
##
## $k
##  [1]      -Inf  7.894521  8.498480  9.125827  9.596461 10.958370 11.497486
##  [8] 12.647860 13.171537 13.951401 16.922844
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Based on the output, the tree size that yields the lowest deviance is 1. However, a tree with a single terminal node does not make sense. The next "best" tree size is 2, which is quite clearly an underfitted tree that is far too simple. Despite having the lowest deviances trees of these sizes are expected to perform poorly on unseen data. The low deviances are most probably due to cross-validation finding such simple trees working well and overfitting on the training data. The optimal tree size, thus, cannot be selected based on the lowest deviance alone.

The tree size with the next lowest deviance is 7. This is chosen, as it is a reasonable tree size that is not too large or small, and still yields a relatively low deviance. `prune.tree()` is used to prune the tree.

```
waus_tree_pruned <- prune.tree(waus_tree, best = 7)
plot(waus_tree_pruned)
text(waus_tree_pruned, pretty = 0)
```



The resulting decision tree is simple and easy to interpret. Consisting of only 7 terminal nodes, it splits on only 4 attributes, compared to the original decision tree which has 22 terminal nodes and splits on 9 attributes.

Similar code from Question 5 is used to evaluate the performance of this pruned decision tree on the test data, create a confusion matrix and report its accuracy.

```
waus_tree_pruned_acc
```

```
## [1] 0.5634518
```

```
waus_tree_pruned_cm
```

```
##                 Actual Class
## Predicted Class  0  1
##               0 58 55
##               1 31 53
```

The accuracy of this pruned decision tree is 0.5634518, which is a slight improvement of that of the original tree (0.5532995).

Similar code from Question 6 is then used to update the plot of ROC curves and compute the AUC value (see Appendix).

```
waus_acc_auc <- rbind(waus_acc_auc, data.frame(model = "Pruned decision tree",
                                               accuracy = waus_tree_pruned_acc,
                                               auc = waus_tree_pruned_auc))
waus_acc_auc
```

```
##                     model  accuracy       auc
## 1           Decision tree 0.5532995 0.5761548
## 2 Naive Bayes classifier 0.5482234 0.5527466
## 3                Bagging 0.6345178 0.6255202
## 4               Boosting 0.6345178 0.6351436
## 5           Random forest 0.5939086 0.6069496
## 6   Pruned decision tree 0.5634518 0.5838535
```

Again, a slight improvement of the AUC value from the original decision tree's 0.5761548 to the pruned tree's 0.5838535 is observed. Most of this improvement is seen at low to medium-high thresholds. When the threshold is high, the pruned tree noticeably performs worse than random guessing.

Question 10

In my attempts to create the best tree-based classifier, I have tried adjusting parameters, cross-validation, and scaling features on the classifiers created in Question 4. Unfortunately, no resulting classifier was able to beat the performance of the boosting and bagging classifiers built in Question 4 with default parameters. Instead, the strongest tree-based classifier I could create is a gradient-boosted trees model (note: I have sought confirmation from Dr Betts that a new type of tree-based classifier can be used for this question).

Gradient boosting is an ensemble method that iteratively trains decision trees to to minimise the errors made by the ensemble of previously-trained trees. The functions to construct a gradient-boosted trees model is provided by the `lightgbm` package.

Gradient boosting only accepts numerical input features and targets. Hence, the factor variables in the training and testing data are first encoded into numerical values.

```
waus_train_lgbm <- waus_train
waus_test_lgbm <- waus_test

for (col in colnames(waus_train_lgbm)[1:21]) {
  if (is.factor(waus_train_lgbm[, col])) {
    waus_train_lgbm[, col] <- as.numeric(waus_train_lgbm[, col])
    waus_test_lgbm[, col] <- as.numeric(waus_test_lgbm[, col])
  }
}

waus_train_lgbm$MHT <- as.numeric(waus_train_lgbm$MHT) - 1
waus_test_lgbm$MHT <- as.numeric(waus_test_lgbm$MHT) - 1
```

The function that fits the model is `lgb.train()`, which requires the input data to be a of a special class called `lgb.Dataset`. The following code adapts the training data to this class.

```
waus_train_lgbmd <- lgb.Dataset(data = as.matrix(waus_train_lgbm[1:21]),
                                label = as.matrix(waus_train_lgbm$MHT))
```

The following parameters were used to get the best-performing gradient-boosted trees model, which is first assigned to a variable to be passed into `lgb.train()`. The parameter values were selected based on repetitive attempts with different values to get the best-performing model.

- `objective = "binary"`: sets `lgb.train()` to train a binary classfication model
- `learning_rate = 0.05`: sets the shrinkage rate (controls how much each tree in the ensemble contributes to the final prediction) to 0.05
- `feature_fraction = 0.7`: sets `lgb.train()` to randomly select 70% of features on each tree node
- `bagging_fraction = 0.8`: sets `lgb.train()` to select 80% of data without resampling for bagging
- `bagging_freq = 10`: sets `lgb.train()` to perform bagging at every 10 iterations

```
params <- list(objective = "binary", learning_rate = 0.05, feature_fraction = 0.7,
               bagging_fraction = 0.8, bagging_freq = 10)
```

`lgb.train()` is then run with these parameters and 100 rounds.

```
waus_lgbm <- lgb.train(params = params, data = waus_train_lgbmd, nrounds = 100)
```

The confusion matrix is created and the accuracy is reported.

```
waus_lgbm_acc
```

```
## [1] 0.6548223
```

```
waus_lgbm_cm
```

```
##                Actual Class
## Predicted Class  0  1
##               0 53 32
##               1 36 76
```

The accuracy of the gradient-boosted trees model is 0.6548223, which outperforms every tree-based classifier we have thus far. As in previous questions, the ROC curve for this model is added to the plot and the AUC value is computed (see Appendix).

```
waus_acc_auc
```

```
##                    model  accuracy       auc
## 1          Decision tree 0.5532995 0.5761548
## 2 Naive Bayes classifier 0.5482234 0.5527466
## 3                Bagging 0.6345178 0.6255202
## 4               Boosting 0.6345178 0.6351436
## 5          Random forest 0.5939086 0.6069496
## 6   Pruned decision tree 0.5634518 0.5838535
## 7 Gradient-boosted trees 0.6548223 0.6369122
```

The AUC value is 0.6369122, which is also higher than that of every tree-based classifier thus far. Based on the ROC curve, this model performs worse than bagging and boosting at medium-high thresholds, but better at all other thresholds. These metrics show that this classifier is indeed better than the others for predictions of MHT.

The reason why gradient boosting was chosen when none of the other tree-based classifiers were able to improve was because gradient boosting is a variant of boosting, which is the most powerful classifier in Question 4. By implementing a possibly better variant of it that tends less to overfit and is good at capturing relationships between predictors, an improvement in perforamance was expected. Gradient boosting has also

been known for strong performance in binary classification tasks in general. All original attributes of `WAUS` were retained as including all of them seems to produce the best model for bagging and boosting. Of course, the factor predictors and target have to be first encoded to numerical values.

Question 11

To implement an artificial neural network (ANN) classifier, the `neuralnet` package is used.

```
library(neuralnet)
```

As ANNs only accept numeric inputs (and only predict numeric outputs), some pre-processing is required. First, all factor attributes (which were originally strings) are dropped from the data frame. This gives a remainder of 17 predictors, which is still enough to construct a reasonably good model. The `MHT` column, which was converted to factors to construct the models in Question 4, is converted back to numeric as well. All the predictors are also normalised using `scale()`. Skipping this step may result in the ANN algorithm not converging before the maximum number of iterations is reached.

```
waus_train_nn <- waus_train[, c(1, 2, 3, 4, 5, 6, 7, 9, 12, 13, 14, 15, 16, 17, 18, 19,
                                21, 22)]
waus_test_nn <- waus_test[, c(1, 2, 3, 4, 5, 6, 7, 9, 12, 13, 14, 15, 16, 17, 18, 19, 21,
                              22)]

waus_train_nn$MHT <- as.numeric(waus_train_nn$MHT) - 1
waus_test_nn$MHT <- as.numeric(waus_test_nn$MHT) - 1

waus_train_nn[1:17] <- scale(waus_train_nn[1:17])
waus_test_nn[1:17] <- scale(waus_test_nn[1:17])
```

Now, the pre-processed data can be used to construct the ANN using `neuralnet()`. As in Question 5, the test data is used for predictions. A confusion matrix is drawn and the accuracy is reported.

```
waus_nn_acc
```

```
## [1] 0.6091371
```

```
waus_nn_cm
```

```
##                Actual Class
## Predicted Class  0  1
##               0 47 35
##               1 42 73
```

The accuracy of the ANN is 0.6091371, which joins bagging, boosting and gradient-boosted trees as a classifier with an accuracy over 0.6.

To get the ROC curve and AUC value of the ANN, the `neuralnet` package has to be first detached as it overrides the `ROCR` package's `prediction()` method.

```
detach(package:neuralnet, unload = TRUE)
```

As the predictions of the ANN are in the form of continuous values between 0 and 1, they effectively represent probabilities of a "1" classification as well. Thus, `waus_nn_predict` is passed directly into `prediction()`. The plot of ROC curves is updated and the AUC value is computed (see Appendix).

```
waus_acc_auc
```

```
##                       model  accuracy       auc
## 1             Decision tree 0.5532995 0.5761548
## 2    Naive Bayes classifier 0.5482234 0.5527466
## 3                   Bagging 0.6345178 0.6255202
```

```
## 4                  Boosting 0.6345178 0.6351436
## 5             Random forest 0.5939086 0.6069496
## 6      Pruned decision tree 0.5634518 0.5838535
## 7    Gradient-boosted trees 0.6548223 0.6369122
## 8 Artificial neural network 0.6091371 0.6852892
```

The AUC value is 0.6852892, which is the best among all the classifiers we have thus far. The ROC curve shows that the ANN is the strongest classifier at high and medium-high thresholds, before gradually being outperformed by gradient-boosted trees. The reason why the AUC value is the highest among all classifiers but the accuracy is not relatively high, may be due to the ANN predicting positives (`MHT = 1`) well but not negatives, as shown by the confusion matrix. Having a high true positive rate contributes to the ANN's higher AUC, but due to mediocre performance on the negative class, the overall accuracy of the model is relatively medium.

Question 12

The new classifier is a support vector machine (SVM) implemented using the R package `kernlab`.

An SVM is a supervised learning model that is applicable for both classification and regression, making it a suitable model for this assignment's data. In binary classification, the SVM attempts to find a hyperplane, or decision boundary, that separates the training data, putting each instance into one of two classes. During training, the hyperplane is formed based on the selection of support vectors, which are the instances from each class that are closest to each other in the feature space. The aim is to maximise the distance between these points and the hyperplane, forming as distinct a separation as possible. SVMs can also implement a kernel function, which transforms the input into a higher-dimensional space. A hyperplane of a higher dimension can thus be found, which may be better at separating the instances.

The method provided by `kernlab` that fits the SVM model is `ksvm()`. The kernel used is the linear kernel, represented by the argument `vanilladot`. This was chosen due to the large number of predictors, which decreases the separability of the data points in a higher dimensional space.

```
waus_svm <- ksvm(MHT ~ ., data = waus_train, kernel = "vanilladot")
```

The confusion matrix is then constructed and the accuracy is reported.

```
waus_svm_acc
```

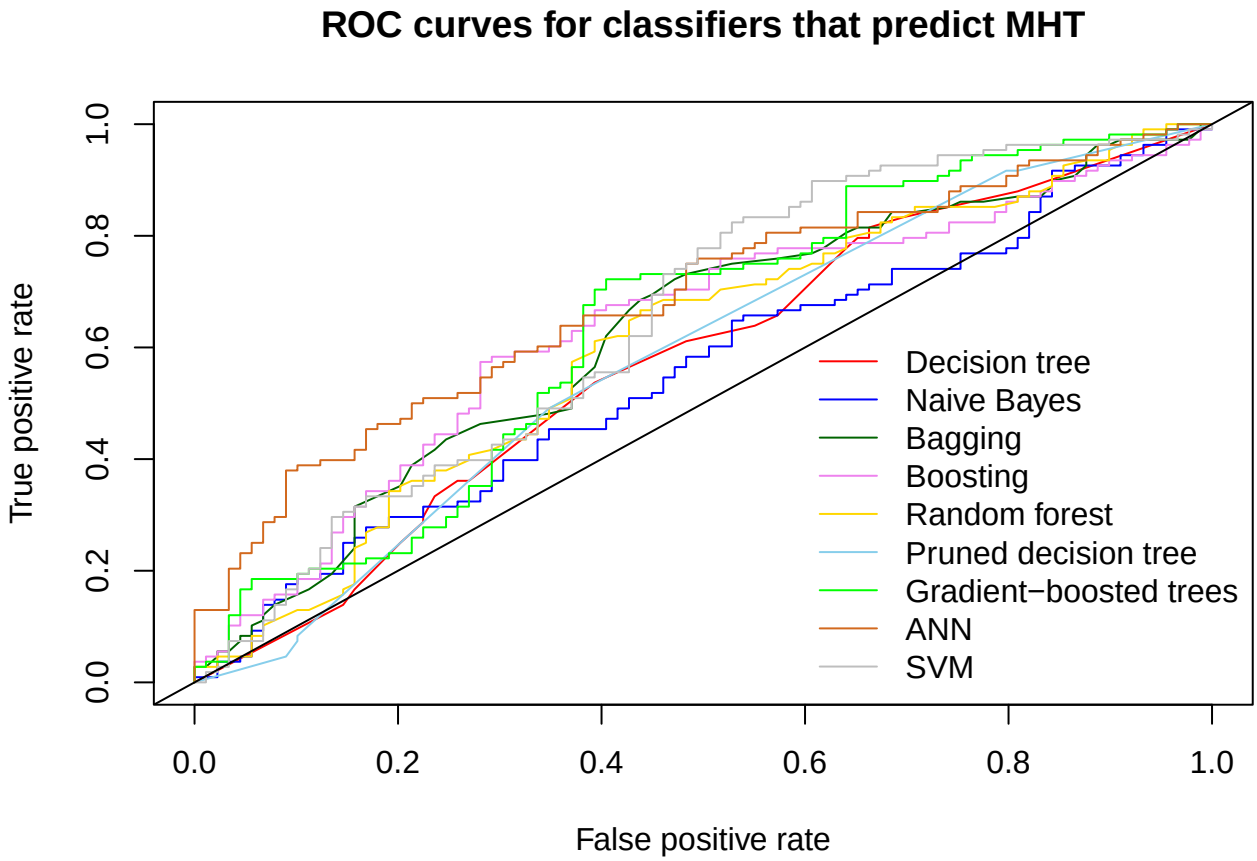
```
## [1] 0.6395939
```

```
waus_svm_cm
```

```
##                 Actual Class
## Predicted Class  0  1
##               0 45 27
##               1 44 81
```

The accuracy of the model is 0.6395939, which is better than that of all classifiers except gradient-boosted trees.

To get the ROC curve and AUC value of the model, `ksvm()` is re-executed to include the `prob.model = TRUE` argument, which gives us the confidence values of whether it is more humid the next day. The same process for the earlier classifiers is then repeated (see Appendix). An updated plot of ROC curves is drawn.




```
waus_acc_auc
```

```
##                       model  accuracy       auc
## 1             Decision tree 0.5532995 0.5761548
## 2    Naive Bayes classifier 0.5482234 0.5527466
## 3                   Bagging 0.6345178 0.6255202
## 4                  Boosting 0.6345178 0.6351436
## 5             Random forest 0.5939086 0.6069496
## 6      Pruned decision tree 0.5634518 0.5838535
## 7    Gradient-boosted trees 0.6548223 0.6369122
## 8 Artificial neural network 0.6091371 0.6852892
## 9    Support vector machine 0.6395939 0.6493966
```

The AUC value is 0.6493966, which exceeds that of all classifiers except the artificial neural network. Based on the ROC curve, this classifier outperforms all others at low thresholds to medium-low thresholds. This tells us that the support vector machine classifier is a good option to use for this data.

Appendix

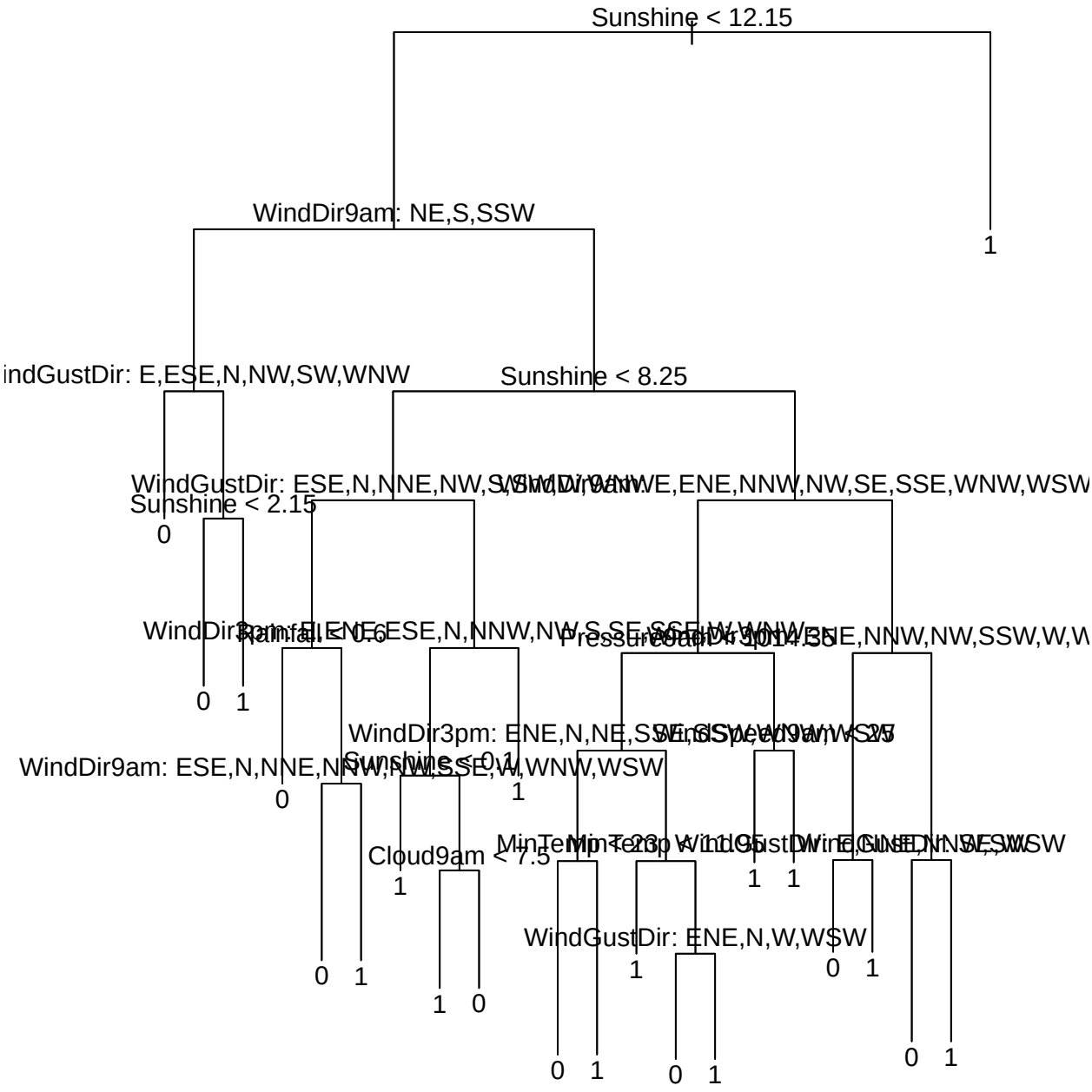
Output of `summary(WAUS)` in Question 1.

```
##       Year          Location       MinTemp          MaxTemp     
##  Min.   :2008   Min.   : 6.00   Min.   :-5.80   Min.   :-3.20  
##  1st Qu.:2011   1st Qu.:14.00   1st Qu.: 6.90   1st Qu.:16.40  
##  Median :2014   Median :24.00   Median :12.10   Median :21.15  
##  Mean   :2014   Mean   :24.46   Mean   :12.25   Mean   :21.78  
##  3rd Qu.:2017   3rd Qu.:38.00   3rd Qu.:17.70   3rd Qu.:27.32  
##  Max.   :2019   Max.   :42.00   Max.   :30.10   Max.   :45.30  
```

```
##  NA's   :22                    NA's   :29      NA's   :28
##     Rainfall        Evaporation        Sunshine       WindGustDir
##  Min.   :  0.000   Min.   : 0.000   Min.   : 0.000   NW     : 160
##  1st Qu.:  0.000   1st Qu.: 2.800   1st Qu.: 4.325   W      : 156
##  Median :  0.000   Median : 4.400   Median : 8.000   E      : 138
##  Mean   :  2.941   Mean   : 4.818   Mean   : 7.154   WNW    : 133
##  3rd Qu.:  1.200   3rd Qu.: 6.600   3rd Qu.:10.200   NNW    : 125
##  Max.   :156.400   Max.   :30.200   Max.   :13.700   (Other):1158
##  NA's   :34        NA's   :726      NA's   :842      NA's   : 130
##  WindGustSpeed      WindDir9am     WindDir3pm    WindSpeed9am    WindSpeed3pm
##  Min.   : 11.00   W      : 175   WNW    : 171   Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 33.00   NNW    : 158   SE     : 157   1st Qu.: 9.00   1st Qu.:13.00
##  Median : 41.00   N      : 146   NW     : 148   Median :15.00   Median :20.00
##  Mean   : 43.62   WNW    : 145   W      : 143   Mean   :15.55   Mean   :19.81
##  3rd Qu.: 52.00   NW     : 144   E      : 142   3rd Qu.:20.00   3rd Qu.:24.00
##  Max.   :130.00   (Other):1097   (Other):1197   Max.   :52.00   Max.   :59.00
##  NA's   :129      NA's   : 135   NA's   :  42   NA's   :37      NA's   :48
##   Pressure9am      Pressure3pm      Cloud9am       Cloud3pm        Temp9am
##  Min.   : 979.1   Min.   : 979   Min.   :0.00   Min.   :0.000   Min.   :-5.00
##  1st Qu.:1011.8   1st Qu.:1009   1st Qu.:2.00   1st Qu.:2.000   1st Qu.:10.80
##  Median :1016.4   Median :1014   Median :6.00   Median :6.000   Median :16.20
##  Mean   :1016.7   Mean   :1014   Mean   :4.84   Mean   :4.842   Mean   :16.24
##  3rd Qu.:1022.0   3rd Qu.:1020   3rd Qu.:7.00   3rd Qu.:7.000   3rd Qu.:21.50
##  Max.   :1040.5   Max.   :1039   Max.   :8.00   Max.   :8.000   Max.   :39.30
##  NA's   :241      NA's   :244    NA's   :816    NA's   :834     NA's   :51
##     Temp3pm      RainToday      RISK_MM            MHT
##  Min.   :-3.60   No  :1442   Min.   :  0.000   Min.   :0.0000
##  1st Qu.:15.20   Yes : 517   1st Qu.:  0.000   1st Qu.:0.0000
##  Median :19.80   NA's:  41   Median :  0.000   Median :1.0000
##  Mean   :20.33               Mean   :  2.823   Mean   :0.5066
##  3rd Qu.:25.70               3rd Qu.:  1.200   3rd Qu.:1.0000
##  Max.   :44.00               Max.   :155.200   Max.   :1.0000
##  NA's   :40                  NA's   :39        NA's   :109
```

Diagram of initial decision tree plotted in Question 4.

```
plot(waus_tree)
text(waus_tree, pretty = 0)
```

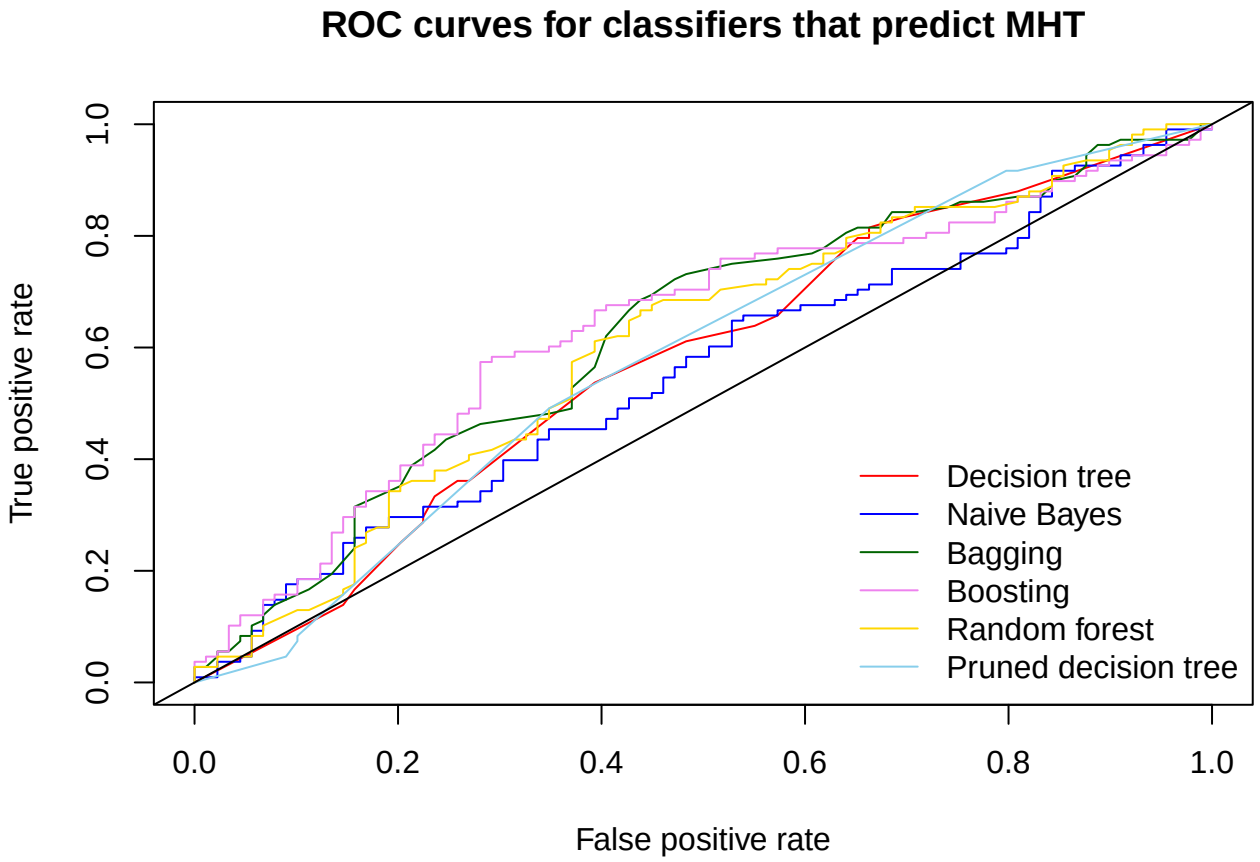


Code to make predictions, create confusion matrix, report accuracy, construct ROC curve and compute AUC value for the simple classifier (pruned decision tree) in Question 9.

```
waus_tree_pruned_predict <- predict(waus_tree_pruned, waus_test, type = "class")
waus_tree_pruned_cm <- table("Predicted Class" = waus_tree_pruned_predict,
                             "Actual Class" = waus_test$MHT)
waus_tree_pruned_acc <- get_accuracy(waus_tree_pruned_cm)

waus_tree_pruned_predict_prob <- predict(waus_tree_pruned, waus_test, type = "vector")
waus_tree_pruned_pred <- prediction(waus_tree_pruned_predict_prob[, 2],  waus_test$MHT)
waus_tree_pruned_perf <- performance(waus_tree_pruned_pred, "tpr", "fpr")

plot(waus_tree_perf, col = "red")
plot(waus_bayes_perf, col = "blue", add = TRUE)
plot(waus_bag_perf, col = "darkgreen", add = TRUE)
```

```
plot(waus_boost_perf, col = "violet", add = TRUE)
plot(waus_forest_perf, col = "gold", add = TRUE)
plot(waus_tree_pruned_perf, col = "skyblue", add = TRUE)

abline(0, 1)
legend("bottomright",
       c("Decision tree", "Naive Bayes", "Bagging", "Boosting", "Random forest",
         "Pruned decision tree"),
       col = c("red", "blue", "darkgreen", "violet", "gold", "skyblue"),
       lty = 1, bty = "n", inset = c(0, 0))
title("ROC curves for classifiers that predict MHT")

waus_tree_pruned_auc <- performance(waus_tree_pruned_pred, "auc")@y.values[[1]]
```

ROC curves at Queestion 9.

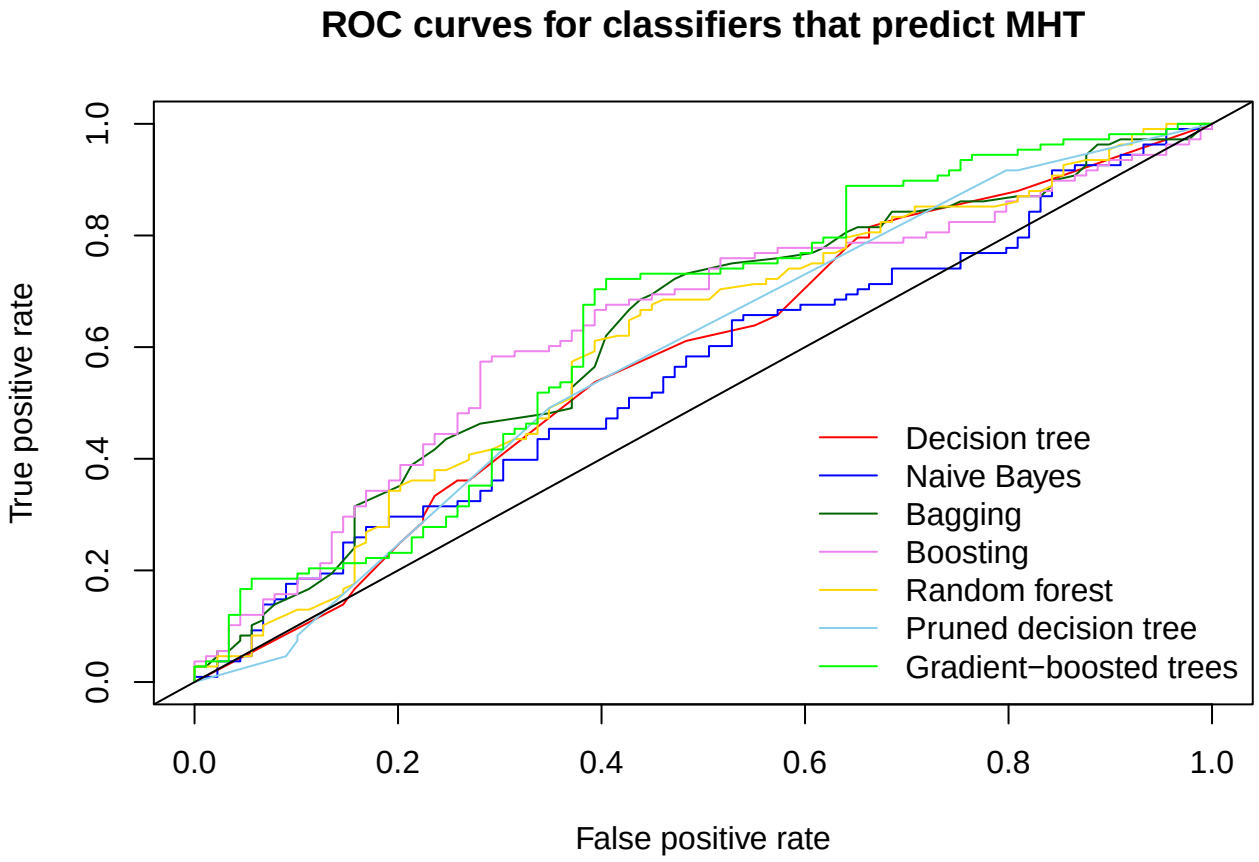


Code to make predictions, create confusion matrix, report accuracy, construct ROC curve, compute AUC value and update classifier comparison table for the best tree-based classifier (gradient-boosted trees) in Question 10.

```
waus_lgbm_predict <- predict(waus_lgbm, as.matrix(waus_test_lgbm[1:21]),
                             type = "response")
waus_lgbm_cm <- table("Predicted Class" = ifelse(waus_lgbm_predict > 0.5, 1, 0),
                      "Actual Class" = waus_test_lgbm$MHT)
waus_lgbm_acc <- get_accuracy(waus_lgbm_cm)
waus_lgbm_pred <- prediction(waus_lgbm_predict, waus_test_lgbm$MHT)
waus_lgbm_perf <- performance(waus_lgbm_pred, "tpr", "fpr")
```

```
plot(waus_tree_perf, col = "red")
plot(waus_bayes_perf, col = "blue", add = TRUE)
plot(waus_bag_perf, col = "darkgreen", add = TRUE)
plot(waus_boost_perf, col = "violet", add = TRUE)
plot(waus_forest_perf, col = "gold", add = TRUE)
plot(waus_tree_pruned_perf, col = "skyblue", add = TRUE)
plot(waus_lgbm_perf, col = "green", add = TRUE)

abline(0, 1)
legend("bottomright",
       c("Decision tree", "Naive Bayes", "Bagging", "Boosting", "Random forest",
         "Pruned decision tree", "Gradient-boosted trees"),
       col = c("red", "blue", "darkgreen", "violet", "gold", "skyblue", "green"),
       lty = 1, bty = "n", inset = c(0, 0))
title("ROC curves for classifiers that predict MHT")

waus_lgbm_auc <- performance(waus_lgbm_pred, "auc")@y.values[[1]]
waus_acc_auc <- rbind(waus_acc_auc,
                      data.frame(model = "Gradient-boosted trees",
                                 accuracy = waus_lgbm_acc, auc = waus_lgbm_auc))
```

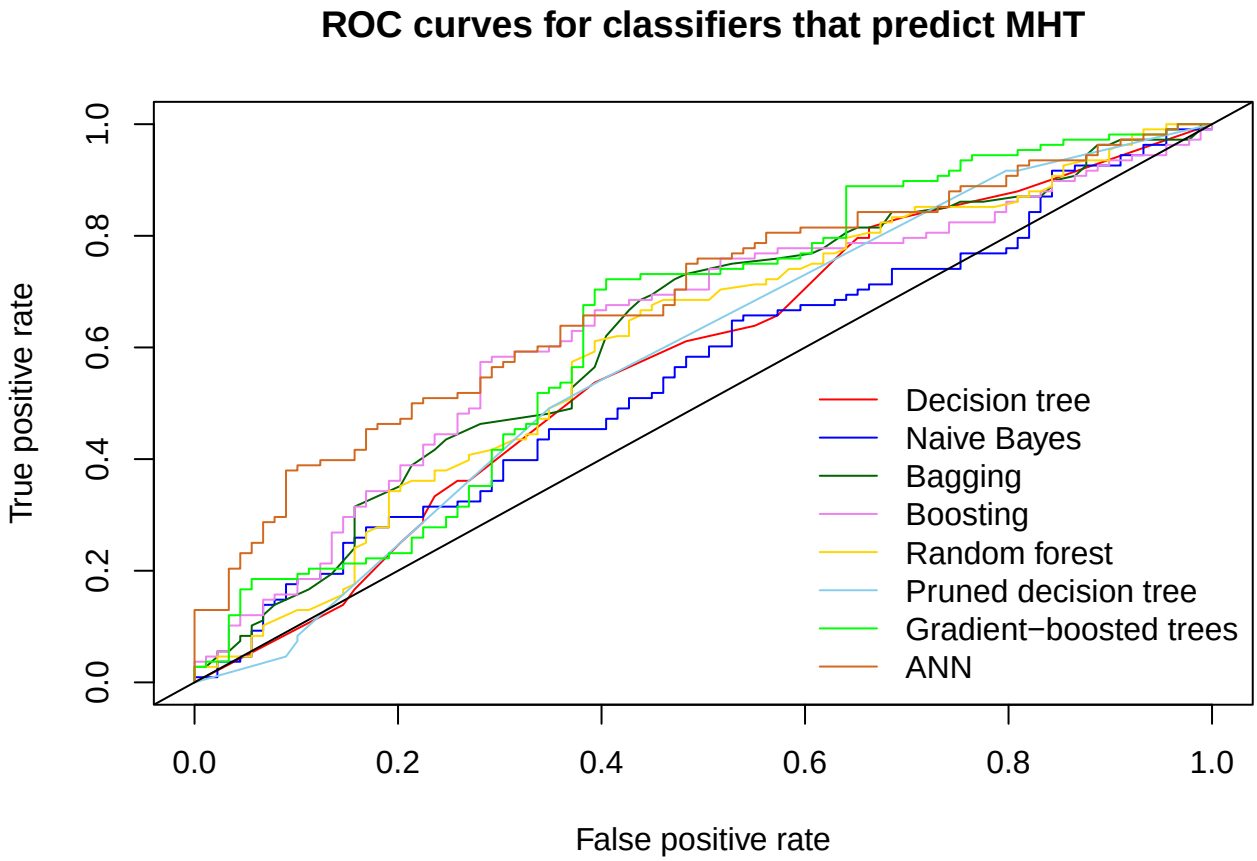
ROC curves at Queestion 10.



Code to make predictions, create confusion matrix, report accuracy, construct ROC curve, compute AUC value and update classifier comparison table for the artificial neural network in Question 11.

```
waus_nn <- neuralnet(MHT == 1 ~ ., waus_train_nn, hidden = 3, linear.output = FALSE)
waus_nn_predict <- predict(waus_nn, waus_test_nn[, 1:17])
waus_nn_cm <- table("Predicted Class" = ifelse(waus_nn_predict > 0.5, 1, 0),
                    "Actual Class" = waus_test_nn$MHT)
waus_nn_acc <- get_accuracy(waus_nn_cm)
waus_nn_pred <- prediction(waus_nn_predict, waus_test_nn$MHT)
waus_nn_perf <- performance(waus_nn_pred, "tpr", "fpr")

plot(waus_tree_perf, col = "red")
plot(waus_bayes_perf, col = "blue", add = TRUE)
plot(waus_bag_perf, col = "darkgreen", add = TRUE)
plot(waus_boost_perf, col = "violet", add = TRUE)
plot(waus_forest_perf, col = "gold", add = TRUE)
plot(waus_tree_pruned_perf, col = "skyblue", add = TRUE)
plot(waus_lgbm_perf, col = "green", add = TRUE)
plot(waus_nn_perf, col = "chocolate", add = TRUE)

abline(0, 1)
legend("bottomright",
       c("Decision tree", "Naive Bayes", "Bagging", "Boosting", "Random forest",
         "Pruned decision tree", "Gradient-boosted trees", "ANN"),
       col = c("red", "blue", "darkgreen", "violet", "gold", "skyblue", "green", "chocolate"),
       lty = 1, bty = "n", inset = c(0, 0))
title("ROC curves for classifiers that predict MHT")

waus_nn_auc <- performance(waus_nn_pred, "auc")@y.values[[1]]
waus_acc_auc <- rbind(waus_acc_auc,
                      data.frame(model = "Artificial neural network",
                                 accuracy = waus_nn_acc, auc = waus_nn_auc))
```

ROC curves at Queestion 11.



Code to make predictions, create confusion matrix, report accuracy, construct ROC curve, compute AUC value and update classifier comparison table for the new classifier (support vector machine) in Question 12.

```
waus_svm_predict <- predict(waus_svm, waus_test)
waus_svm_cm <- table("Predicted Class" = waus_svm_predict,
                     "Actual Class" = waus_test$MHT)
waus_svm_acc <- get_accuracy(waus_svm_cm)

waus_svm_prob <- ksvm(MHT ~ ., data = waus_train, kernel = "vanilladot",
                      prob.model = TRUE)
waus_svm_predict_prob <- predict(waus_svm_prob, waus_test, type = "prob")
waus_svm_pred <- prediction(waus_svm_predict_prob[, 2], waus_test$MHT)
waus_svm_perf <- performance(waus_svm_pred, "tpr", "fpr")

plot(waus_tree_perf, col = "red")
plot(waus_bayes_perf, col = "blue", add = TRUE)
plot(waus_bag_perf, col = "darkgreen", add = TRUE)
plot(waus_boost_perf, col = "violet", add = TRUE)
plot(waus_forest_perf, col = "gold", add = TRUE)
plot(waus_tree_pruned_perf, col = "skyblue", add = TRUE)
plot(waus_lgbm_perf, col = "green", add = TRUE)
plot(waus_nn_perf, col = "chocolate", add = TRUE)
plot(waus_svm_perf, col = "grey", add = TRUE)

abline(0, 1)
legend("bottomright",
       c("Decision tree", "Naive Bayes", "Bagging", "Boosting", "Random forest",
```

```
         "Pruned decision tree", "Gradient-boosted trees", "ANN", "SVM"),
       col = c("red", "blue", "darkgreen", "violet", "gold", "skyblue", "green",
               "chocolate", "grey"),
       lty = 1, bty = "n", inset = c(0, 0))
title("ROC curves for classifiers that predict MHT")

waus_svm_auc <- performance(waus_svm_pred, "auc")@y.values[[1]]
waus_acc_auc <- rbind(waus_acc_auc, data.frame(model = "Support vector machine",
                                               accuracy = waus_svm_acc,
                                               auc = waus_svm_auc))
```