STAT 218 - Analytics Project II

Inigo Benavides and Rommel Bartolome

March 13, 2019

Abstract

We created two tree-based models using a dataset collection of 300 sampled water districts in the Philippines. The first model is a regression tree model with water prices as the output variable while the second one is a categorical tree model where we created an output variable called wastage rating. A Pruned Decision Tree, a Random Forest and a Gradient Boosted Decision Tree was employed. It has been found that the best model for regression is the Random Forest at 0.246 RMSE. The Pruned Decision Tree on the other hand has been found to be the best model for classification with an optimal AUC of 0.535 and 56% Accuracy.

Introduction

In this project, we were given a dataset of 300 sampled water districts in the Philippines. The specific locations of the water districts have been anonymised and no reference year is provided. There was no autocorrelation, and we will assume that there would be no spatial correlation between districts.

Using this data, we will be creating two models using tree-based methods. The first model would be a regression tree model with the water prices as output variable while the second model would be a categorical tree model where we created a new output variable called `wastage rating`. For the `wastage rating`, if the percent of non-revenue water from total displaced water (`nrwpercent`) is less than or equal 25, we label it as 1 and 0 otherwise.

We will be employing several tree-based modelling methods. First, we will try a simple Decision Tree and prune it accordingly. Then, we will use a Random Forest in an effort to improve our prediction. Lastly, we will employ boosting using Gradient Boosted Decision Trees.

Data Loading and Cleaning

Before creating our models, we first load all the libraries we will be using in this project:

```
library("tidyverse")
library("caret")
library("GGally")
library("car")
library("pROC")
library("randomForest")
library("gbm")
library("rpart")
library("rpart.plot")
```

We will now clean our data and set our seed for reproducibility. Here, we factorize necessary variables and based on previous work, we transform and take the logarithm of `conn` (number of connections in a water district), `vol_nrw` (volume of non-revenue water in cu.m., which is displaced water in which the water district did not collect revenues) and `wd_rate` (water rate in pesos for a specific water district, as minimum charge for the first 10 cu. m.). We also simplify `Mun1` (number of first-class municipalities in the water district) as a binary decision while `conn_p_area` (number of connections per square kilometre) was squared. Lastly, the wastage rating which we will call as `nrwpcent_class` is added for the classfication model.

```
set.seed(1)
df <- read_csv("data_BaBe.csv") %>%
```

```r
  select(-c(X1)) %>% #Remove insignificant column
  mutate(REGION=as.factor(REGION),
         WD.Area=as.factor(WD.Area),
         Mun1=as.factor(case_when(Mun1 > 0 ~ 1, TRUE ~ 0)),
         conn_log=log(conn),
         vol_nrw_log=log(vol_nrw),
         wd_rate_log=log(wd_rate),
         conn_p_area_squared=conn_p_area^2,
         nrwpcent_class=as.factor(case_when(nrwpcent <= 25 ~ 1, TRUE ~ 0))
         # Engineer target classification variable
         )
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   REGION = col_character(),
##   WD.Area = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

The data were then split to a train and a test dataset.

```r
# Train test split first 250 vs. last 50
df_train <- df[1:250,]
df_test <- df[251:300,]
df %>% head
```

```
## # A tibble: 6 x 24
##   REGION WD.Area  conn conn_p_area wd_rate vol_nrw nrwpcent cities Mun1
##   <fct>  <fct>   <dbl>       <dbl>   <dbl>   <dbl>    <dbl>  <dbl> <fct>
## 1 IV     Area 3   2330       22.7     184    51125       26      0 0
## 2 VI     Area 5  23390        6.08    392. 1408272       31      1 0
## 3 V      Area 4   5268       15.8     368.  368233       40      1 0
## 4 I      Area 1   3255       30.9     283   473606       56      0 1
## 5 II     Area 1   1420       22.3     198.   34465       20      0 0
## 6 IX     Area 9   3484       16.1     348.  208936       36      1 0
## # ... with 15 more variables: Mun2 <dbl>, Mun3 <dbl>, Mun4 <dbl>,
## #   Mun5 <dbl>, gw <dbl>, sprw <dbl>, surw <dbl>, elevar <dbl>,
## #   coastal <dbl>, emp <dbl>, conn_log <dbl>, vol_nrw_log <dbl>,
## #   wd_rate_log <dbl>, conn_p_area_squared <dbl>, nrwpcent_class <fct>
```

In the following sections, we will explore the fitting of the following models to our water district data set: (1) Decision Tree with Pruning, (2) Random Forest, and (3) Gradient Boosted Decision Trees. The first part will be for the Regression Tree Model while the latter parts will be for the Categorical Tree Model.

Regression Tree Models

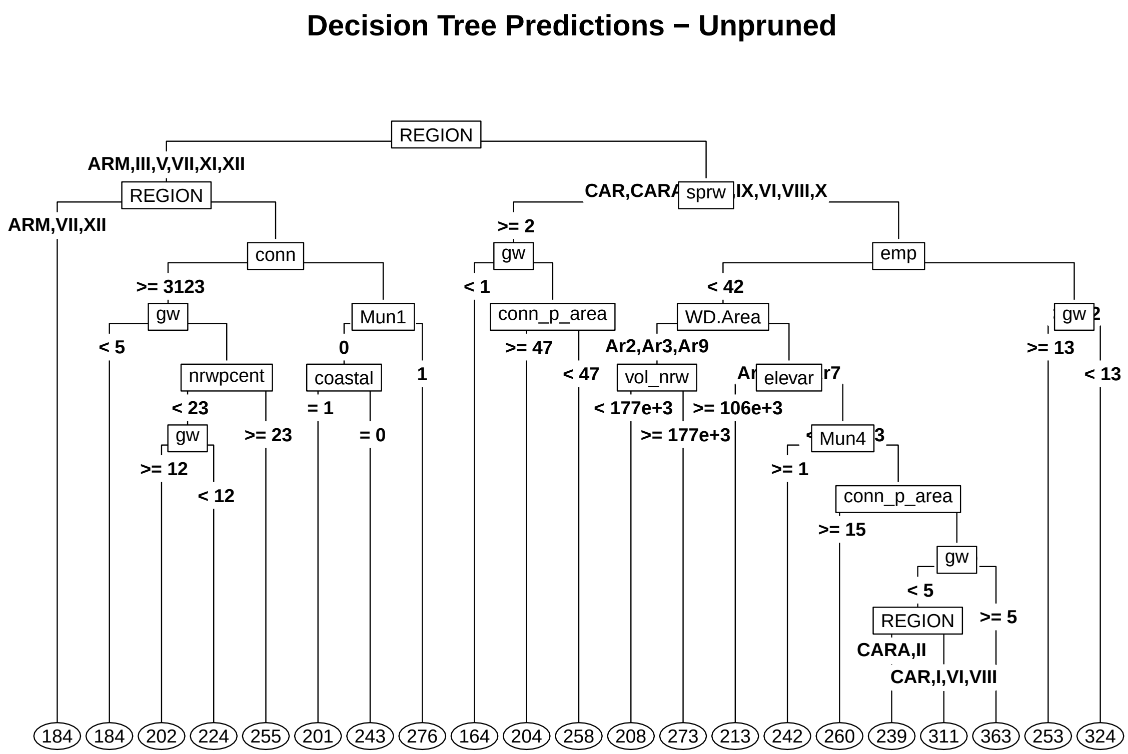
Regression Decision Tree

We grow first a simple basic unpruned tree:

```r
tree_model_simple <- df_train %>%
  rpart(wd_rate ~ REGION + WD.Area + conn + conn_p_area + vol_nrw + nrwpcent + cities +
          Mun1 + Mun2 + Mun3 + Mun4 + Mun5 + gw + sprw + surw + elevar + coastal + emp
        , data=., control=rpart.control(cp=0), model = TRUE)
```

We visualize this tree:

```r
prp(tree_model_simple, varlen = 100, type=5,
    split.round = .5,
    cex=0.6,
    fallen.leaves = TRUE,
    main="Decision Tree Predictions - Unpruned"
    )
```



Here, using all variables, we can see that this is a very “bushy” tree. We would like to check the effectiveness of this unpruned tree but before anything else, we will first create a helper function called `evaluateRMSE` that will help us evaluate the RMSE of the models:

```r
evaluateRMSE <- function(model, df_set) {
  predictions <- model %>% predict(df_set) %>% as.vector()
  obs <- df_set$wd_rate %>% as.vector()
  rmse <- sqrt(mean((predictions - obs)^2)) / (mean(obs))
  return(rmse)
}
```

Evaluating our initial unpruned tree,

```r
evaluateRMSE(tree_model_simple, df_test)
```

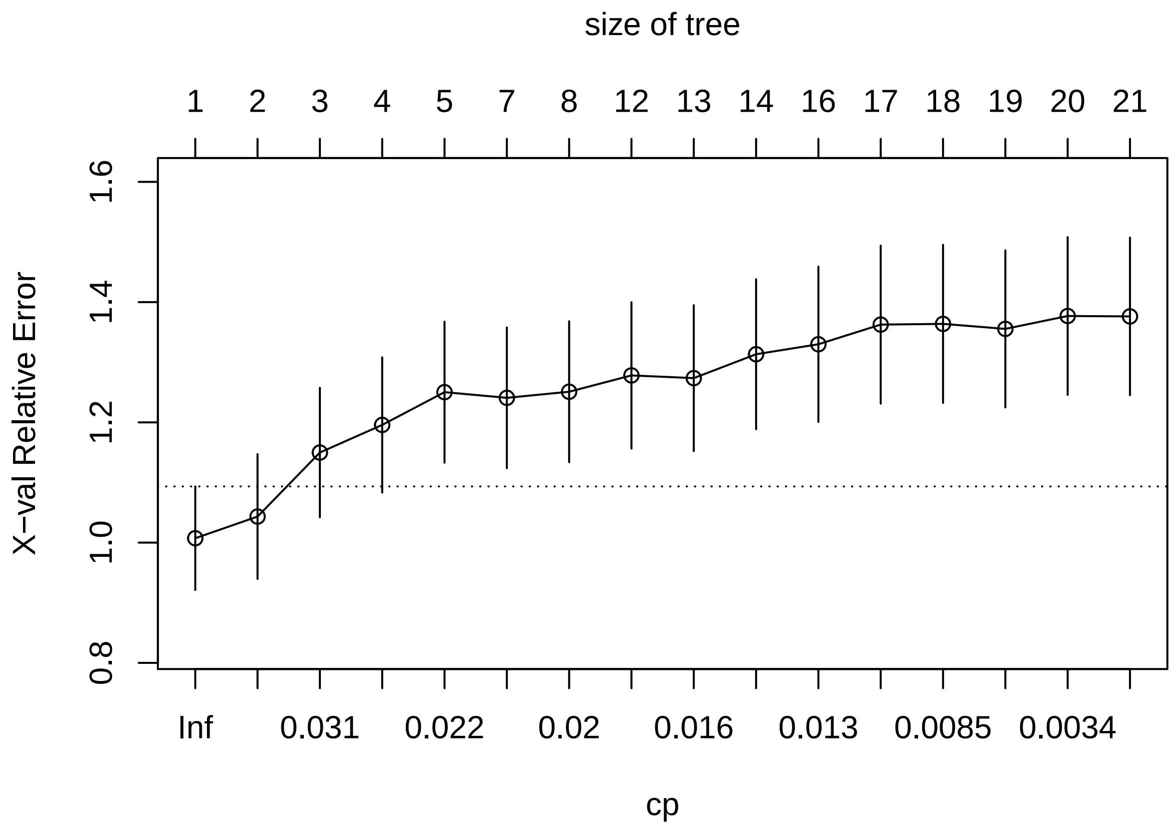
```
## [1] 0.3424502
```

We get an RMSE of 0.342 using the basic unpruned tree. We prune it down by observing the behaviour of the complexity parameter (cp).

```r
printcp(tree_model_simple)
```

```
##
## Regression tree:
## rpart(formula = wd_rate ~ REGION + WD.Area + conn + conn_p_area +
##     vol_nrw + nrwpcent + cities + Mun1 + Mun2 + Mun3 + Mun4 +
##     Mun5 + gw + sprw + surw + elevar + coastal + emp, data = .,
##     model = TRUE, control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
##  [1] coastal     conn        conn_p_area elevar      emp
##  [6] gw          Mun1        Mun4        nrwpcent    REGION
## [11] sprw        vol_nrw     WD.Area
##
## Root node error: 1338891/250 = 5355.6
##
## n= 250
##
##           CP nsplit rel error xerror     xstd
## 1  0.0756057      0   1.00000 1.0073 0.086127
## 2  0.0340840      1   0.92439 1.0433 0.103813
## 3  0.0287991      2   0.89031 1.1498 0.107732
## 4  0.0239481      3   0.86151 1.1958 0.112475
## 5  0.0210538      4   0.83756 1.2502 0.117304
## 6  0.0204349      6   0.79546 1.2408 0.117111
## 7  0.0191697      7   0.77502 1.2509 0.117249
## 8  0.0173964     11   0.69618 1.2781 0.121774
## 9  0.0152727     12   0.67878 1.2735 0.121342
## 10 0.0151980     13   0.66351 1.3133 0.124703
## 11 0.0118906     15   0.63311 1.3299 0.129251
## 12 0.0098337     16   0.62122 1.3626 0.131539
## 13 0.0074012     17   0.61139 1.3638 0.131467
## 14 0.0059240     18   0.60399 1.3555 0.130604
## 15 0.0019106     19   0.59806 1.3769 0.131262
## 16 0.0000000     20   0.59615 1.3762 0.131217
```

```r
plotcp(tree_model_simple)
```



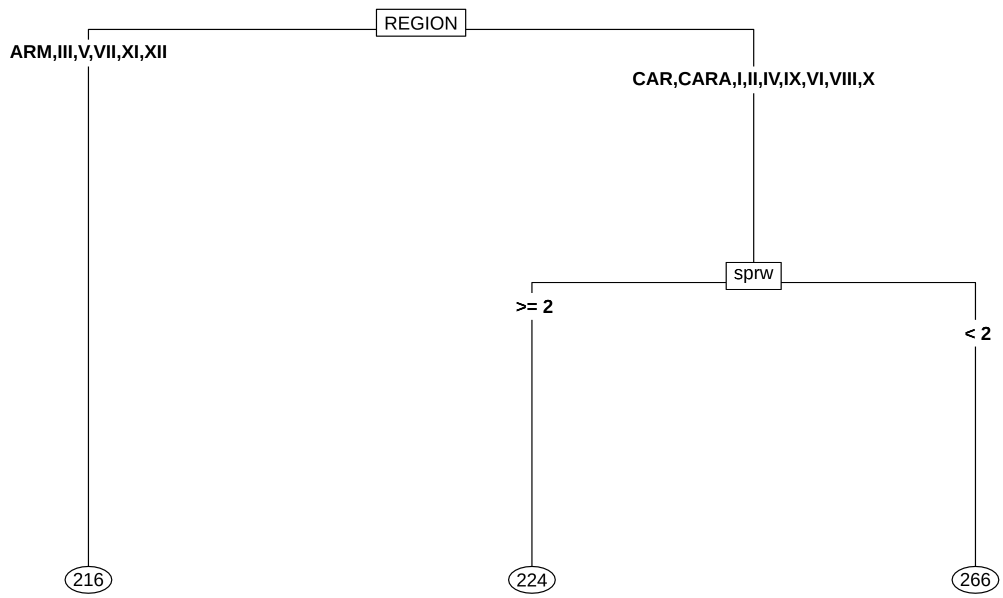
Here, we can see that at `cp = 0.031`, the tree looks optimal. We use it to prune the tree:

```
tree_model <- df_train %>%
  rpart(wd_rate ~ REGION + WD.Area + conn + conn_p_area + vol_nrw + nrwpcent + cities +
          Mun1 + Mun2 + Mun3 + Mun4 + Mun5 + gw + sprw + surw + elevar + coastal + emp
        , data=., control=rpart.control(cp=0.031), model = TRUE)
```

Visualizing our pruned tree:

```
prp(tree_model, varlen = 100, type=5,
    split.round = .5,
    cex=0.6,
    fallen.leaves = TRUE,
    main="Decision Tree Predictions - Pruned"
    )
```

Decision Tree Predictions − Pruned



This looks very simple compared to our initial unpruned tree. We see that a split is made on the region level, separating `ARM`, `III`, `V`, `VII`, `XI`, `XII` with a prediction on the `wd_rate` of 216. Those predicted with regions `CAR`, `CARA`, `I`, `II`, `IV`, `IX`, `VI`, `VIII`, and `X` are then further split depending on `sprw` at a value of 2. We now check the RMSE and check if it has improved:

```
df_test$model_prediction_decision_tree <- tree_model %>% predict(df_test)
evaluateRMSE(tree_model, df_test)
```

```
## [1] 0.2682703
```

By fitting a pruned decision tree model on the training set, with a complexity parameter of 0.031, we get a model with a test RMSE of 0.268, which is quite similar to our full linear regression model's performance of 0.26. However, this is definitely much better than our unpruned decision tree with an RMSE of 0.342.

Random Forest - Regression

In Random Forest, we create lots of trees and then average them to reduce variance. We create our Random Forest using the `randomForest` function:

```
set.seed(1)
random_forest_model <- randomForest(wd_rate ~ REGION + WD.Area + conn +
                                      conn_p_area + vol_nrw + nrwpcent +
                                      cities + Mun1 + Mun2 + Mun3 + Mun4 +
                                      Mun5 + gw + sprw + surw + elevar +
                                      coastal + emp, data=df_train, importance=TRUE)

random_forest_model %>% summary
```

```
##                 Length Class  Mode
## call              4    -none- call
## type              1    -none- character
## predicted       250    -none- numeric
## mse             500    -none- numeric
## rsq             500    -none- numeric
## oob.times       250    -none- numeric
## importance       36    -none- numeric
## importanceSD     18    -none- numeric
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           11    -none- list
## coefs             0    -none- NULL
## y               250    -none- numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

```r
random_forest_model$importance
```

```
##                %IncMSE IncNodePurity
## REGION      161.319032    183176.607
## WD.Area      49.445506    103539.087
## conn        283.363566    100730.257
## conn_p_area 192.863321    141387.638
## vol_nrw     249.866198    109514.749
## nrwpcent     74.540705     99094.401
## cities        3.834686     10238.604
## Mun1         84.495881     22774.232
## Mun2        -12.537939     14859.993
## Mun3        -21.325857      7242.400
## Mun4         71.530505     19858.888
## Mun5          7.198346      4532.094
## gw          471.835717     97896.393
## sprw        177.076989     57786.029
## surw         -5.253585     27483.702
## elevar       83.653815    126289.593
## coastal      20.866034     16766.606
## emp         265.712704     93792.609
```

Fitting a random forest model on the data set, we show the variable importances both in terms of the increase in MSE and increase in node purity. Among the most importance variables by MSE increase are `gw`, `vol_nrw`, `conn` and `conn_p_area`.

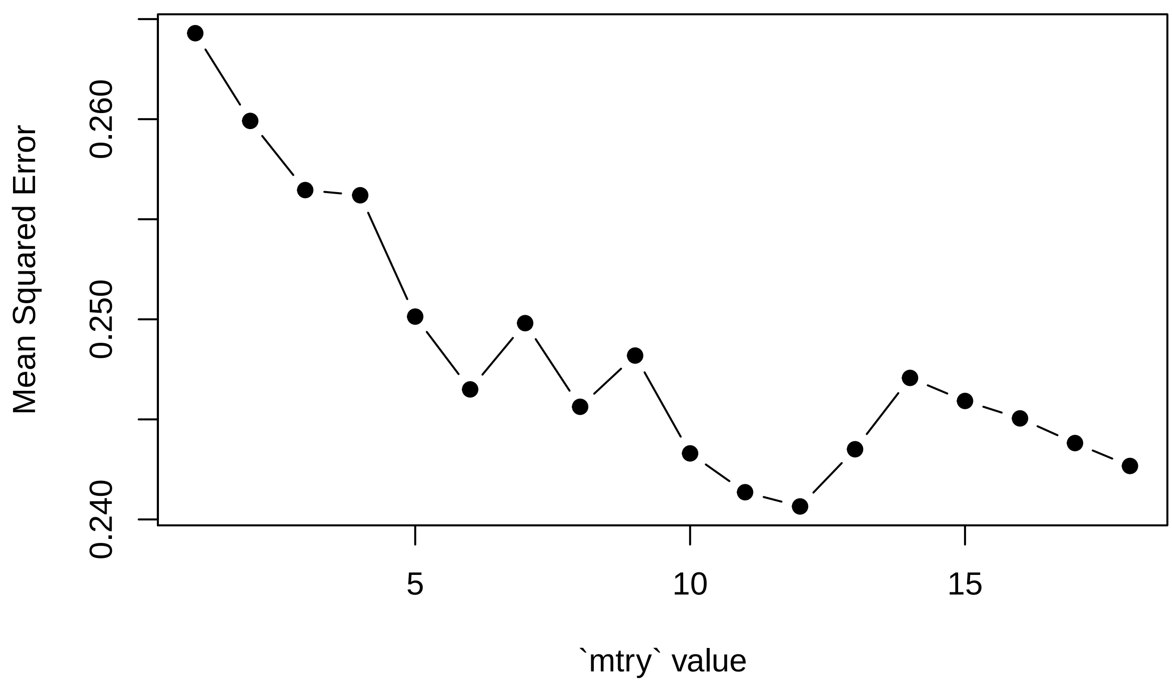
```r
df_test$model_prediction_random_forest <- random_forest_model %>% predict(df_test)
evaluateRMSE(random_forest_model, df_test)
```

```
## [1] 0.2464975
```

Above we find that the random forest outperforms both the linear regression and decision tree model with a test RMSE of 0.246.

The parameters for tuning random forests are quite limited and for this one, we will just use `mtry`. We check the optimum `mtry` value:

```r
test.err <- double(18)
for (mtry in 1:18){
  set.seed(1)
  random_forest_model <- randomForest(wd_rate ~ REGION + WD.Area + conn + conn_p_area +
                                      vol_nrw + nrwpcent + cities + Mun1 + Mun2 + Mun3 +
                                      Mun4 + Mun5 + gw + sprw + surw + elevar + coastal +
                                      emp, data=df_train, importance=TRUE, mtry = mtry)
  test.err[mtry] <- evaluateRMSE(random_forest_model, df_test)
}
matplot(1:mtry, test.err, type = "b", pch = 19, ylab = "Mean Squared Error",
        xlab = "`mtry` value")
```



Here, we see that the most optimum `mtry` value, which is the number of variables randomly chosen at each split is at 12. We try to evaluate our Random Forest at this value:

```r
set.seed(1)
random_forest_model <- randomForest(wd_rate ~ REGION + WD.Area + conn + conn_p_area +
                                    vol_nrw + nrwpcent + cities + Mun1 + Mun2 + Mun3 +
                                    Mun4 + Mun5 + gw + sprw + surw + elevar + coastal +
                                    emp, data=df_train, importance=TRUE, mtry = 12)
evaluateRMSE(random_forest_model, df_test)
```

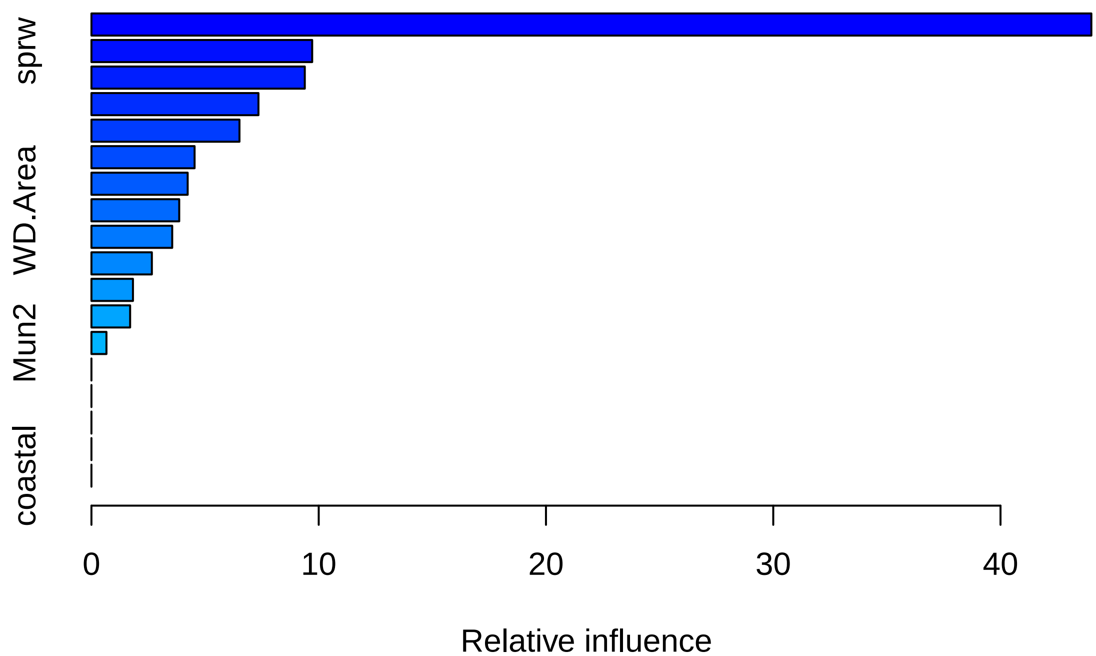
```
## [1] 0.2406504
```

Here, we can see that we achieved a slightly better but quite insignificant decrease in RMSE at 0.241. It seems that using the default `mtry` is sufficient in this model.

Gradient Boosted Regression Trees

Lastly, we use boosting to create regression trees. Boosting tries to patch up the deficiencies of the current ensemble. We create our gradient boosted regression tree:

```
set.seed(1)
gradient_boosted_model <- gbm(wd_rate ~ REGION + WD.Area + conn + conn_p_area +
                                vol_nrw + nrwpcent + cities + Mun1 + Mun2 +
                                Mun3 + Mun4 + Mun5 + gw + sprw + surw + elevar +
                                coastal + emp, data=df_train, distribution="gaussian",
                              cv.folds=5)
gradient_boosted_model %>% summary
```



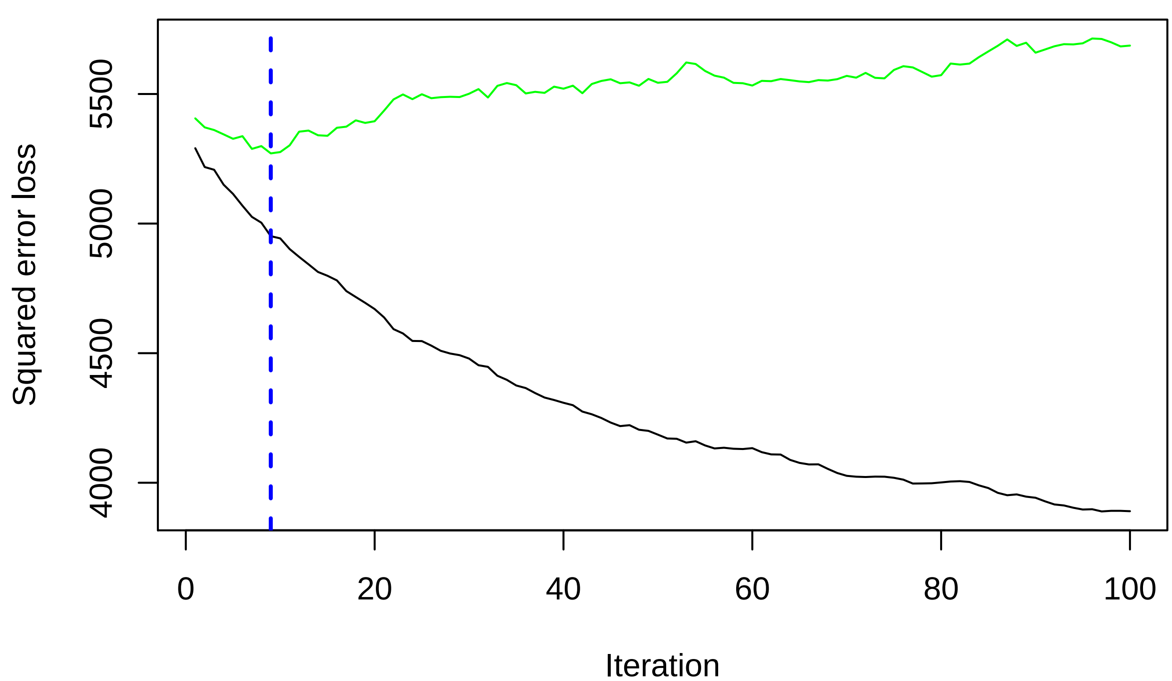

```
##                     var    rel.inf
## REGION           REGION 43.9974447
## sprw               sprw  9.7131378
## conn_p_area conn_p_area  9.3880631
## emp                 emp  7.3492385
## gw                   gw  6.5132895
## elevar           elevar  4.5382881
## vol_nrw         vol_nrw  4.2346542
## WD.Area         WD.Area  3.8631916
## nrwpcent       nrwpcent  3.5544890
## conn               conn  2.6583518
## Mun1               Mun1  1.8273751
## surw               surw  1.7023301
## Mun2               Mun2  0.6601466
```

```
## cities           cities  0.0000000
## Mun3               Mun3  0.0000000
## Mun4               Mun4  0.0000000
## Mun5               Mun5  0.0000000
## coastal         coastal  0.0000000
```

Above, we fit a gradient boosted regression model with 5-fold cross validation on the data set and plot the relative influence of each variable.

Indeed, we find that `REGION` has quite a high relative performance, followed by `conn_p_area` and `elevar`.

```r
# Check performance using 5-fold cross-validation
set.seed(1)
best.iter <- gbm.perf(gradient_boosted_model, method = "cv")
```




```r
print(best.iter)
```

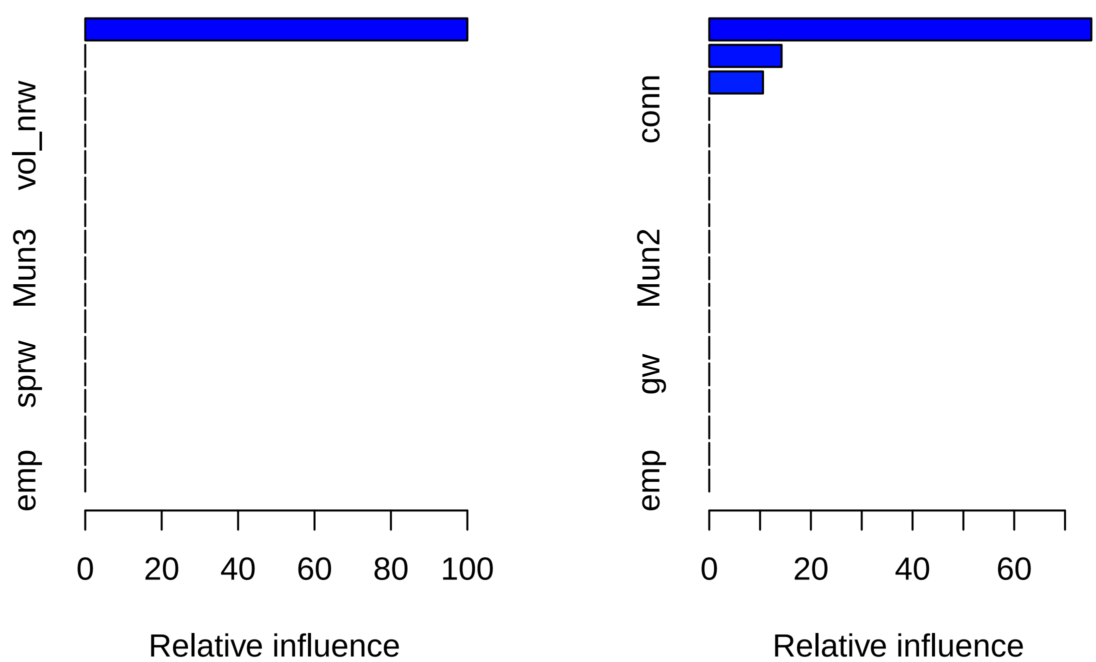
```
## [1] 9
```

```r
# Plot relative influence of each variable
par(mfrow = c(1, 2))
summary(gradient_boosted_model, n.trees = 1)          # using first tree
```

```
##                     var rel.inf
## WD.Area         WD.Area     100
## REGION           REGION       0
## conn               conn       0
## conn_p_area conn_p_area       0
## vol_nrw         vol_nrw       0
```

```
## nrwpcent       nrwpcent       0
## cities           cities       0
## Mun1               Mun1       0
## Mun2               Mun2       0
## Mun3               Mun3       0
## Mun4               Mun4       0
## Mun5               Mun5       0
## gw                   gw       0
## sprw               sprw       0
## surw               surw       0
## elevar           elevar       0
## coastal         coastal       0
## emp                 emp       0
```

```
summary(gradient_boosted_model, n.trees = best.iter)   # using estimated best number of trees
```




```
##                    var  rel.inf
## REGION          REGION 75.18099
## WD.Area        WD.Area 14.23914
## sprw              sprw 10.57987
## conn              conn  0.00000
## conn_p_area conn_p_area  0.00000
## vol_nrw        vol_nrw  0.00000
## nrwpcent      nrwpcent  0.00000
## cities          cities  0.00000
## Mun1              Mun1  0.00000
## Mun2              Mun2  0.00000
```

```
## Mun3                Mun3  0.00000
## Mun4                Mun4  0.00000
## Mun5                Mun5  0.00000
## gw                    gw  0.00000
## surw                surw  0.00000
## elevar            elevar  0.00000
## coastal          coastal  0.00000
## emp                  emp  0.00000
```

If we check the relative performance of the variables between models with the first tree and the one with best number of trees as determined by cross validation, we find that `sprw` was important for the first model, and that `REGION` was again important for the optimal model.

```
Yhat <- predict(gradient_boosted_model, newdata = df_test,
                n.trees = best.iter, type = "link")
obs <- df_test$wd_rate %>% as.vector()
rmse <- sqrt(mean((Yhat - obs)^2)) / (mean(obs))
rmse
```

```
## [1] 0.2610999
```

Interestingly, the GBM model yielded a comparable test RMSE of 0.262 to the pruned decision tree, slightly worse than random forest model.

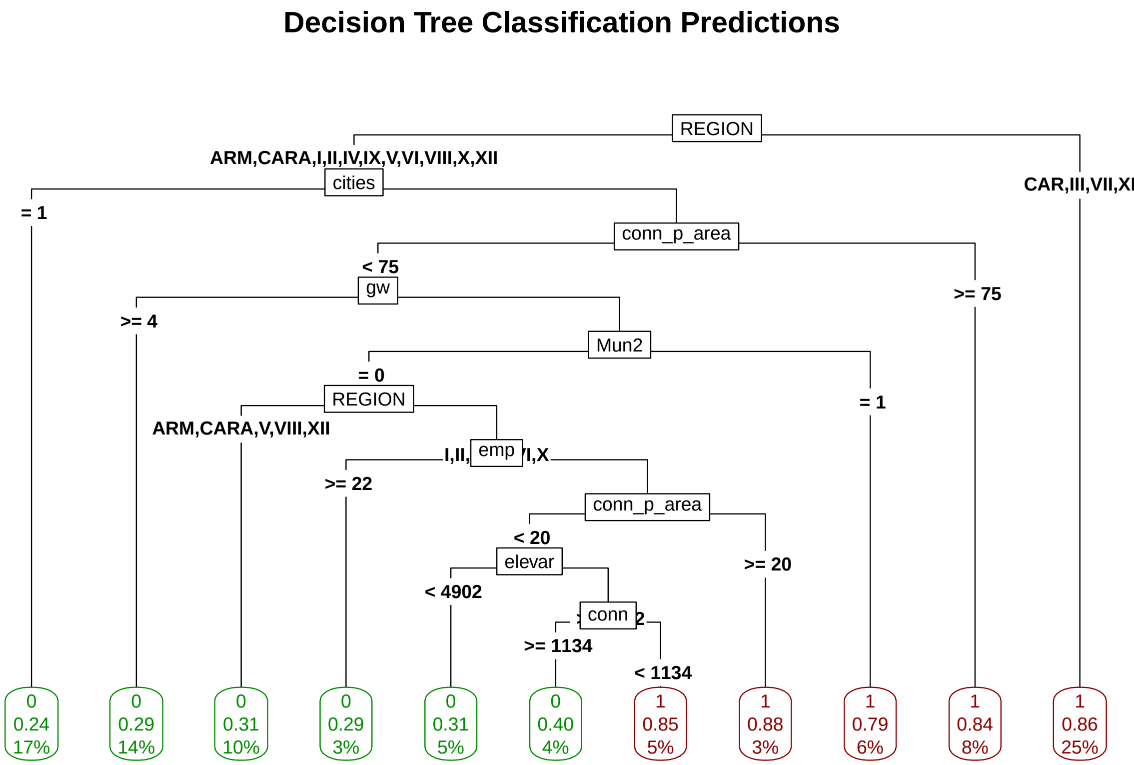
Classification Problem

In this section, we apply the same set of algorithms to the classification problem, taking out `vol_nrw` and `nrwpcent` from the predictors.

Decision Tree

```
set.seed(1)
decision_tree_classifier <- df_train %>%
  mutate(nrwpcent_class=as.factor(nrwpcent_class)) %>%
  rpart(nrwpcent_class ~ REGION + WD.Area + conn + conn_p_area + cities +
          Mun1 + Mun2 + Mun3 + Mun4 + Mun5 + gw + sprw + surw +
          elevar + coastal + emp, data=.,
        control=rpart.control(cp=0.0), model=TRUE)
```

```
cols <- ifelse(decision_tree_classifier$frame$yval == 1, "green4", "darkred")
prp(decision_tree_classifier, varlen = 100, type=5, extra=106,
    split.round = .5,
    col=cols, border.col=cols,
    cex=0.6,
    fallen.leaves = TRUE,
    main="Decision Tree Classification Predictions"
    )
```

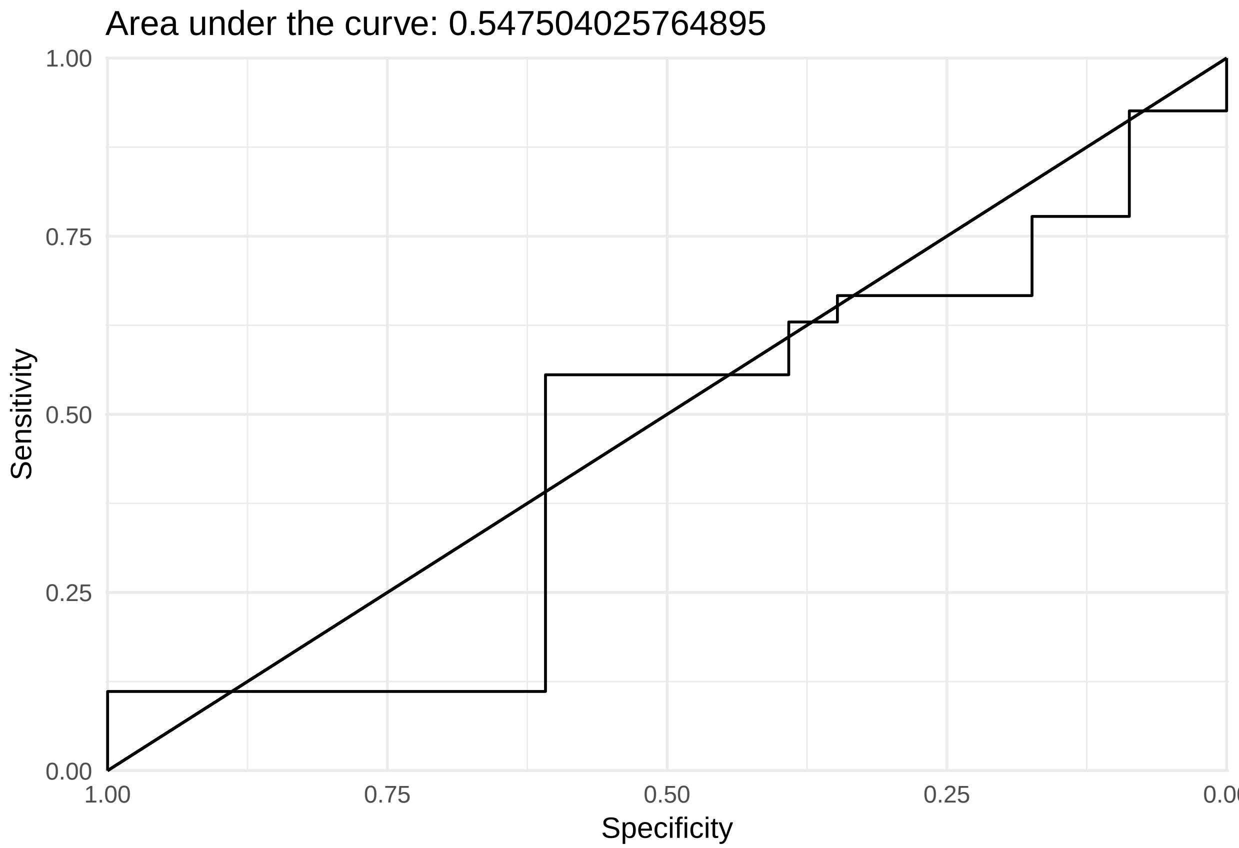


Above, we fit a decision tree model to predict `nrwpcent_class` with no constraint on the complexity, and we find that, again, `REGION` appears to be an important split. Checking its performance:

```
df_test$model_prediction_decision_tree_classifier <-
  predict(decision_tree_classifier, df_test)[,1]
decision_tree_classifier_roc <- roc(df_test$nrwpcent_class,
                                    df_test$model_prediction_decision_tree_classifier)

auc_score <- decision_tree_classifier_roc$auc

cbind(rev(decision_tree_classifier_roc$specificities),
      rev(decision_tree_classifier_roc$sensitivities)) %>%
  as.data.frame() %>%
  rename('Specificity'=V1, 'Sensitivity'=V2) %>%
  ggplot(aes(x=Specificity, y=Sensitivity)) +
  geom_segment(aes(x = 0, y = 1, xend = 1,yend = 0), alpha = 0.5)  +
  geom_step() +
  scale_x_reverse(name = "Specificity",limits = c(1,0), expand = c(0.001,0.001)) +
  scale_y_continuous(name = "Sensitivity", limits = c(0,1), expand = c(0.001, 0.001)) +
  labs(title=paste("Area under the curve:", auc_score, sep=" ")) +
  theme_minimal()
```



The AUC of this model is a marginal 0.547, indicating that it has little predictive power in the test set; indeed, since we removed the complexity parameter constraint, we see that the model severely overfits.

```
# Confusion matrix
predicted_probabilities <- df_test$model_prediction_decision_tree_classifier
predicted_probabilities[predicted_probabilities > 0.5] <- 1
predicted_probabilities[predicted_probabilities <= 0.5] <- 0
predicted_probabilities <- predicted_probabilities %>% as.vector %>% as.factor
observed <- df_test$nrwpcent_class
confusionMatrix(data=predicted_probabilities, reference=observed)
```

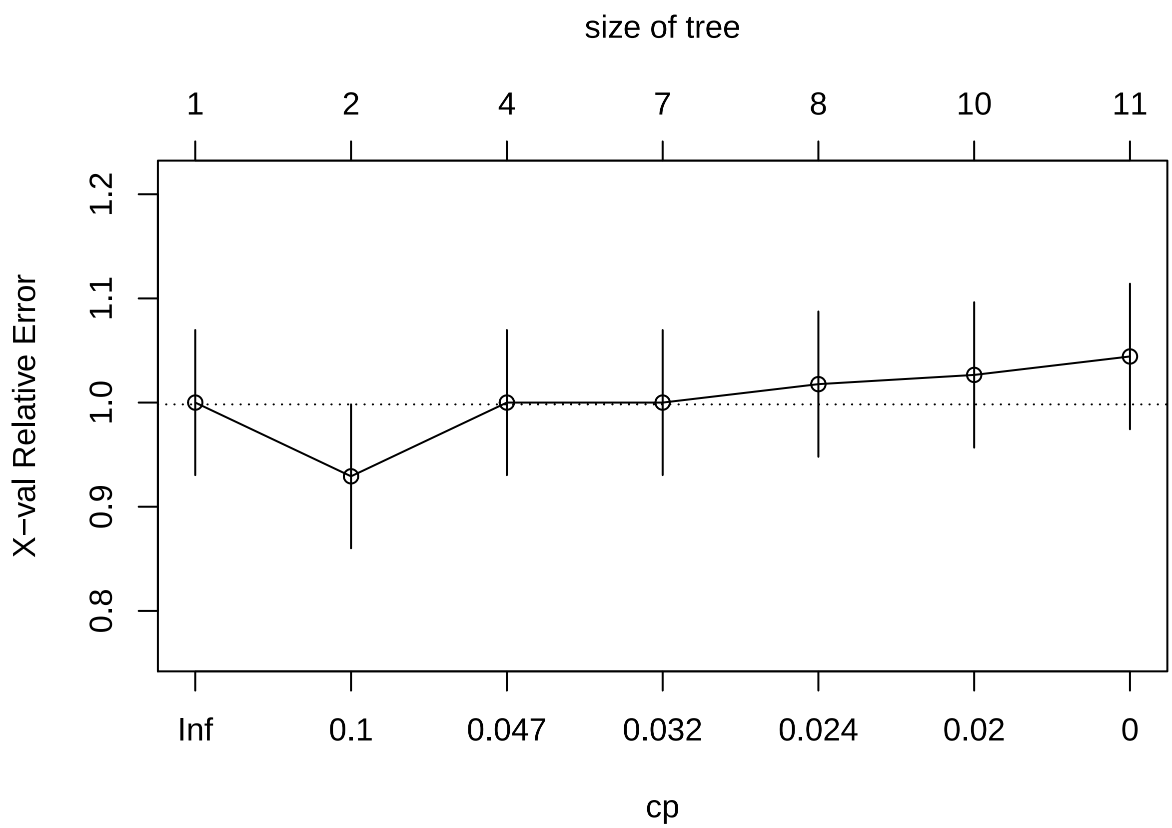
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 14 17
##          1  9 10
##
##                Accuracy : 0.48
##                  95% CI : (0.3366, 0.6258)
##     No Information Rate : 0.54
##     P-Value [Acc > NIR] : 0.8397
##
##                   Kappa : -0.0204
##  Mcnemar's Test P-Value : 0.1698
##
##             Sensitivity : 0.6087
```

```
##             Specificity : 0.3704
##          Pos Pred Value : 0.4516
##          Neg Pred Value : 0.5263
##              Prevalence : 0.4600
##          Detection Rate : 0.2800
##    Detection Prevalence : 0.6200
##       Balanced Accuracy : 0.4895
##
##        'Positive' Class : 0
##
```

By evaluating the model in terms of the test set confusion matrix, we find that this model has an accuracy of 48%, on par with our logistic regression model.

We now try to prune our model. We check the complexity parameter:

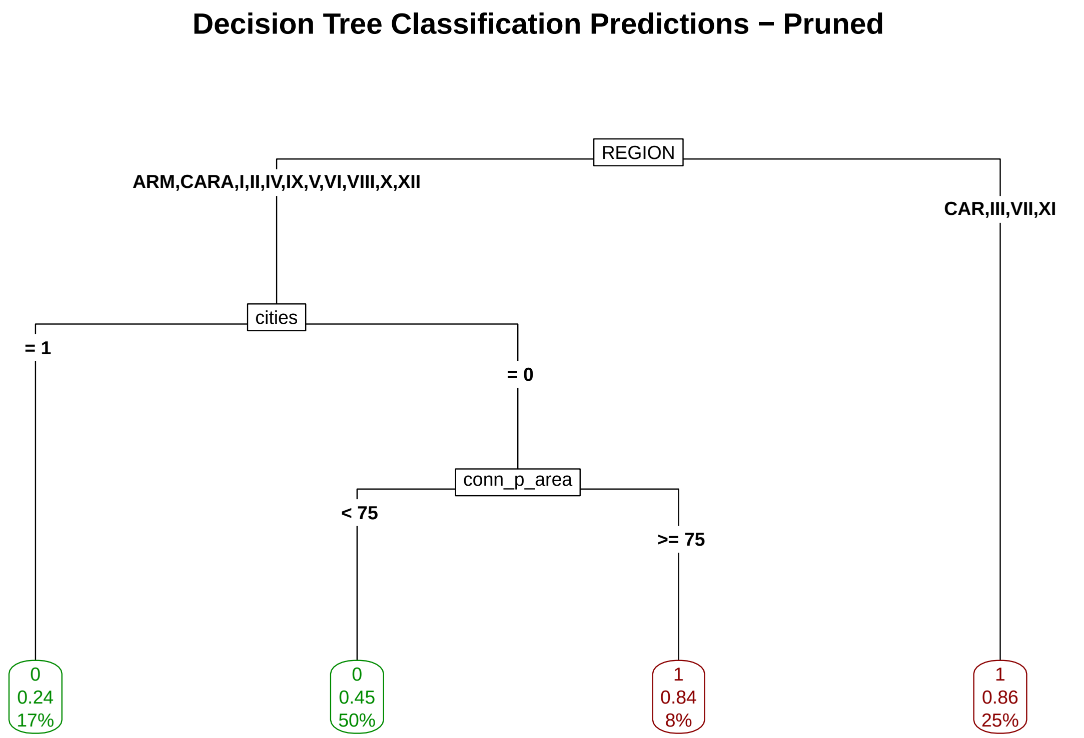
```
plotcp(decision_tree_classifier)
```



Here, it seems that the at `cp = 0.047`, the pruning will be optimal. We do the same process, and see the pruned tree:

```
set.seed(1)
decision_tree_classifier_pruned <- df_train %>%
  mutate(nrwpcent_class=as.factor(nrwpcent_class)) %>%
  rpart(nrwpcent_class ~ REGION + WD.Area + conn + conn_p_area + cities +
          Mun1 + Mun2 + Mun3 + Mun4 + Mun5 + gw + sprw +
          surw + elevar + coastal + emp, data=.,
        control=rpart.control(cp=0.047), model=TRUE)
```

```r
cols <- ifelse(decision_tree_classifier_pruned$frame$yval == 1, "green4", "darkred")
prp(decision_tree_classifier_pruned, varlen = 100, type=5, extra=106,
    split.round = .5,
    col=cols, border.col=cols,
    cex=0.6,
    fallen.leaves = TRUE,
    main="Decision Tree Classification Predictions - Pruned"
    )
```



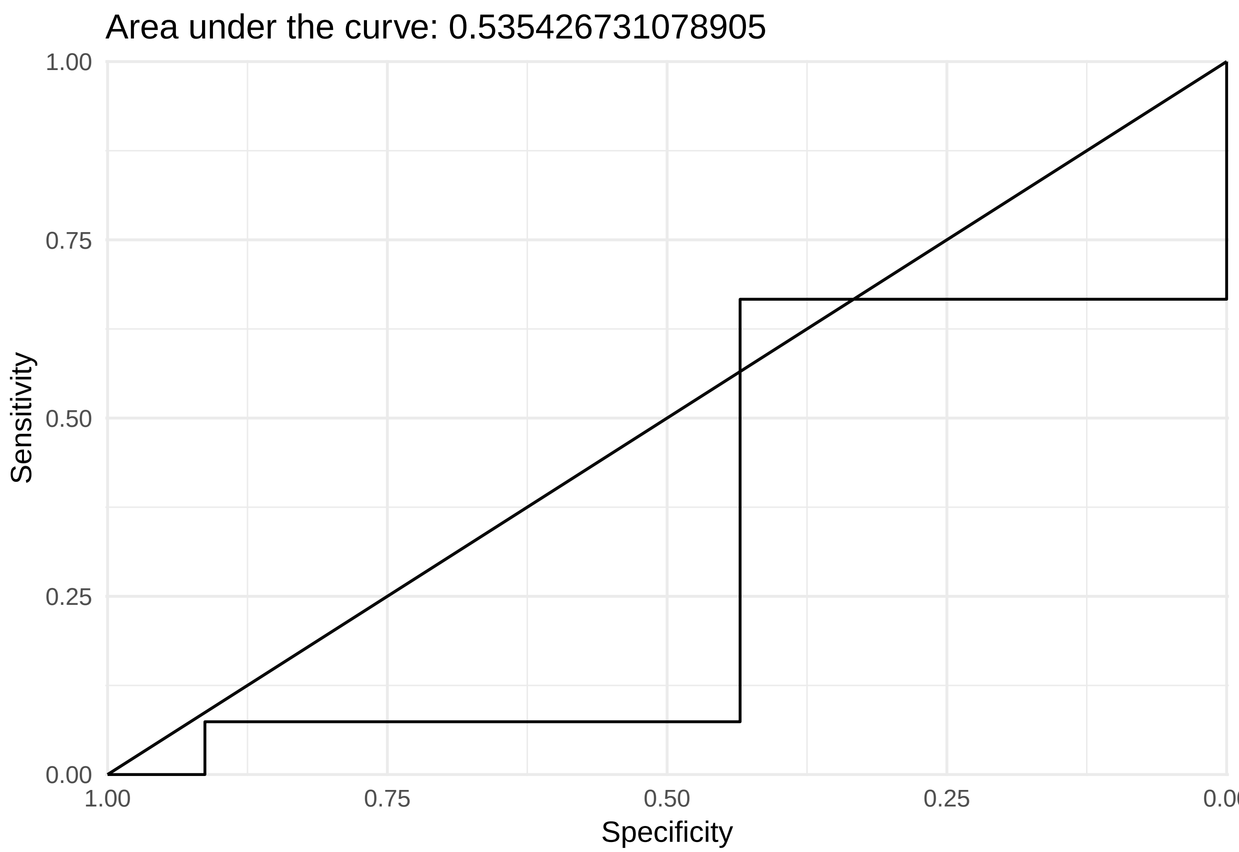
Checking the AUC:

```r
df_test$model_prediction_decision_tree_classifier_pruned <-
  predict(decision_tree_classifier_pruned, df_test)[,1]
decision_tree_classifier_roc_pruned <-
  roc(df_test$nrwpcent_class,
      df_test$model_prediction_decision_tree_classifier_pruned)

auc_score <- decision_tree_classifier_roc_pruned$auc

cbind(rev(decision_tree_classifier_roc_pruned$specificities),
      rev(decision_tree_classifier_roc_pruned$sensitivities)) %>%
  as.data.frame() %>%
  rename('Specificity'=V1, 'Sensitivity'=V2) %>%
  ggplot(aes(x=Specificity, y=Sensitivity)) +
  geom_segment(aes(x = 0, y = 1, xend = 1,yend = 0), alpha = 0.5)  +
  geom_step() +
```

```
  scale_x_reverse(name = "Specificity",limits = c(1,0), expand = c(0.001,0.001)) +
  scale_y_continuous(name = "Sensitivity", limits = c(0,1), expand = c(0.001, 0.001)) +
  labs(title=paste("Area under the curve:", auc_score, sep=" ")) +
  theme_minimal()
```



We achieved a slightly higher AUC of 0.535. We now check the accuracy:

```
# Confusion matrix
predicted_probabilities_pruned <- df_test$model_prediction_decision_tree_classifier_pruned
predicted_probabilities_pruned[predicted_probabilities_pruned > 0.5] <- 1
predicted_probabilities_pruned[predicted_probabilities_pruned <= 0.5] <- 0
predicted_probabilities_pruned <- predicted_probabilities_pruned %>% as.vector %>% as.factor
observed_pruned <- df_test$nrwpcent_class
confusionMatrix(data=predicted_probabilities_pruned, reference=observed_pruned)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 10  9
##          1 13 18
##
##                Accuracy : 0.56
##                  95% CI : (0.4125, 0.7001)
##     No Information Rate : 0.54
##     P-Value [Acc > NIR] : 0.4452
##
```

```
##                  Kappa : 0.1028
##  Mcnemar's Test P-Value : 0.5224
##
##             Sensitivity : 0.4348
##             Specificity : 0.6667
##          Pos Pred Value : 0.5263
##          Neg Pred Value : 0.5806
##              Prevalence : 0.4600
##          Detection Rate : 0.2000
##    Detection Prevalence : 0.3800
##       Balanced Accuracy : 0.5507
##
##        'Positive' Class : 0
##
```

Here, even though we already pruned the decision tree, it still gives us an accuracy of 56%.

Random Forest

```
set.seed(1)
random_forest_classifier <- df_train %>%
  mutate(nrwpcent_class=as.factor(nrwpcent_class)) %>%
  randomForest(nrwpcent_class ~ REGION + WD.Area + conn + conn_p_area +
                 cities + Mun1 + Mun2 + Mun3 + Mun4 + Mun5 + gw + sprw +
                 surw + elevar + coastal + emp, data=., importance=TRUE)

random_forest_classifier %>% summary
```

```
##                 Length Class  Mode
## call               4   -none- call
## type               1   -none- character
## predicted        250   factor numeric
## err.rate        1500   -none- numeric
## confusion          6   -none- numeric
## votes            500   matrix numeric
## oob.times        250   -none- numeric
## classes            2   -none- character
## importance        64   -none- numeric
## importanceSD      48   -none- numeric
## localImportance    0   -none- NULL
## proximity          0   -none- NULL
## ntree              1   -none- numeric
## mtry               1   -none- numeric
## forest            14   -none- list
## y                250   factor numeric
## test               0   -none- NULL
## inbag              0   -none- NULL
## terms              3   terms  call
```

```
random_forest_classifier$importance
```

```
##                         0             1 MeanDecreaseAccuracy
## REGION       4.649346e-02  0.0005223845         2.109209e-02
## WD.Area      3.557196e-02 -0.0020388531         1.485398e-02
## conn         2.145202e-03  0.0046113437         3.243318e-03
```

```
## conn_p_area  6.496103e-03  0.0022144978         4.116597e-03
## cities       9.946280e-03  0.0012673695         5.057683e-03
## Mun1        -2.164387e-03 -0.0007343693        -1.375075e-03
## Mun2         2.521469e-03  0.0006833174         1.471219e-03
## Mun3         1.806716e-03  0.0012378804         1.480989e-03
## Mun4         2.902236e-03  0.0024545085         2.766749e-03
## Mun5         5.593465e-04 -0.0002571943         7.959601e-05
## gw           7.199169e-03 -0.0006427408         2.622378e-03
## sprw        -2.339234e-03  0.0064825809         2.355060e-03
## surw         1.293276e-05 -0.0002186353        -1.279995e-04
## elevar      -7.380035e-03  0.0047994238        -6.492660e-04
## coastal     -1.037109e-03  0.0008958733        -4.152974e-05
## emp          5.490917e-03 -0.0010954709         2.068852e-03
##             MeanDecreaseGini
## REGION             19.779090
## WD.Area            12.670348
## conn               14.834323
## conn_p_area        16.625014
## cities              2.961698
## Mun1                1.802320
## Mun2                2.640596
## Mun3                1.895961
## Mun4                2.141029
## Mun5                1.070545
## gw                  9.339447
## sprw                5.766372
## surw                1.196384
## elevar             14.460307
## coastal             2.488904
## emp                12.368392
```

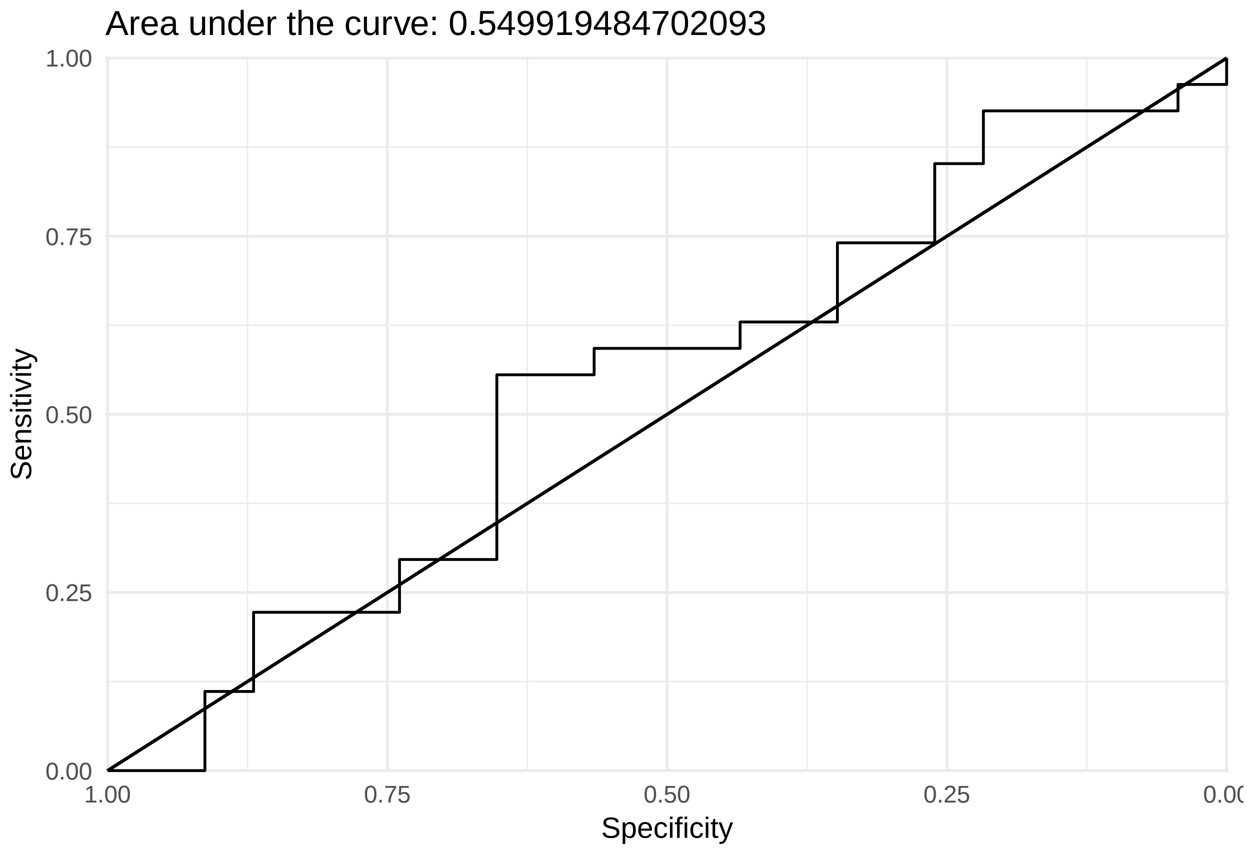
Above, we display the performance of each feature based on the fit random forest model.

We now check its performance:

```
df_test$model_prediction_random_forest_classifier <-
  predict(random_forest_classifier, newdata=df_test, type="prob")[,2]
random_forest_classifier_roc <- roc(df_test$nrwpcent_class,
                                    df_test$model_prediction_random_forest_classifier)

auc_score <- random_forest_classifier_roc$auc

cbind(rev(random_forest_classifier_roc$specificities),
      rev(random_forest_classifier_roc$sensitivities)) %>%
  as.data.frame() %>%
  rename('Specificity'=V1, 'Sensitivity'=V2) %>%
  ggplot(aes(x=Specificity, y=Sensitivity)) +
  geom_segment(aes(x = 0, y = 1, xend = 1,yend = 0), alpha = 0.5)  +
  geom_step() +
  scale_x_reverse(name = "Specificity",limits = c(1,0),
                  expand = c(0.001,0.001)) +
  scale_y_continuous(name = "Sensitivity", limits = c(0,1),
                     expand = c(0.001, 0.001)) +
  labs(title=paste("Area under the curve:", auc_score, sep=" ")) +
  theme_minimal()
```



Interestingly, the random forest appears to perform slighty better than the decision tree model with an AUC of 0.550.

```
# Confusion matrix
predicted_probabilities <- df_test$model_prediction_random_forest_classifier
predicted_probabilities[predicted_probabilities > 0.5] <- 1
predicted_probabilities[predicted_probabilities <= 0.5] <- 0
predicted_probabilities <- predicted_probabilities %>% as.vector %>% as.factor
observed <- df_test$nrwpcent_class
confusionMatrix(data=predicted_probabilities, reference=observed)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  9 10
##          1 14 17
##
##                Accuracy : 0.52
##                  95% CI : (0.3742, 0.6634)
##     No Information Rate : 0.54
##     P-Value [Acc > NIR] : 0.6657
##
##                   Kappa : 0.0212
##  Mcnemar's Test P-Value : 0.5403
##
##             Sensitivity : 0.3913
```

```
##             Specificity : 0.6296
##          Pos Pred Value : 0.4737
##          Neg Pred Value : 0.5484
##              Prevalence : 0.4600
##          Detection Rate : 0.1800
##    Detection Prevalence : 0.3800
##       Balanced Accuracy : 0.5105
##
##        'Positive' Class : 0
##
```

However, if we consider the test set confusion matrix, we find that this model has a lower accuracy of 52%.

Gradient Boosted Classifier

Lastly, we create a gradient boosted classifier:

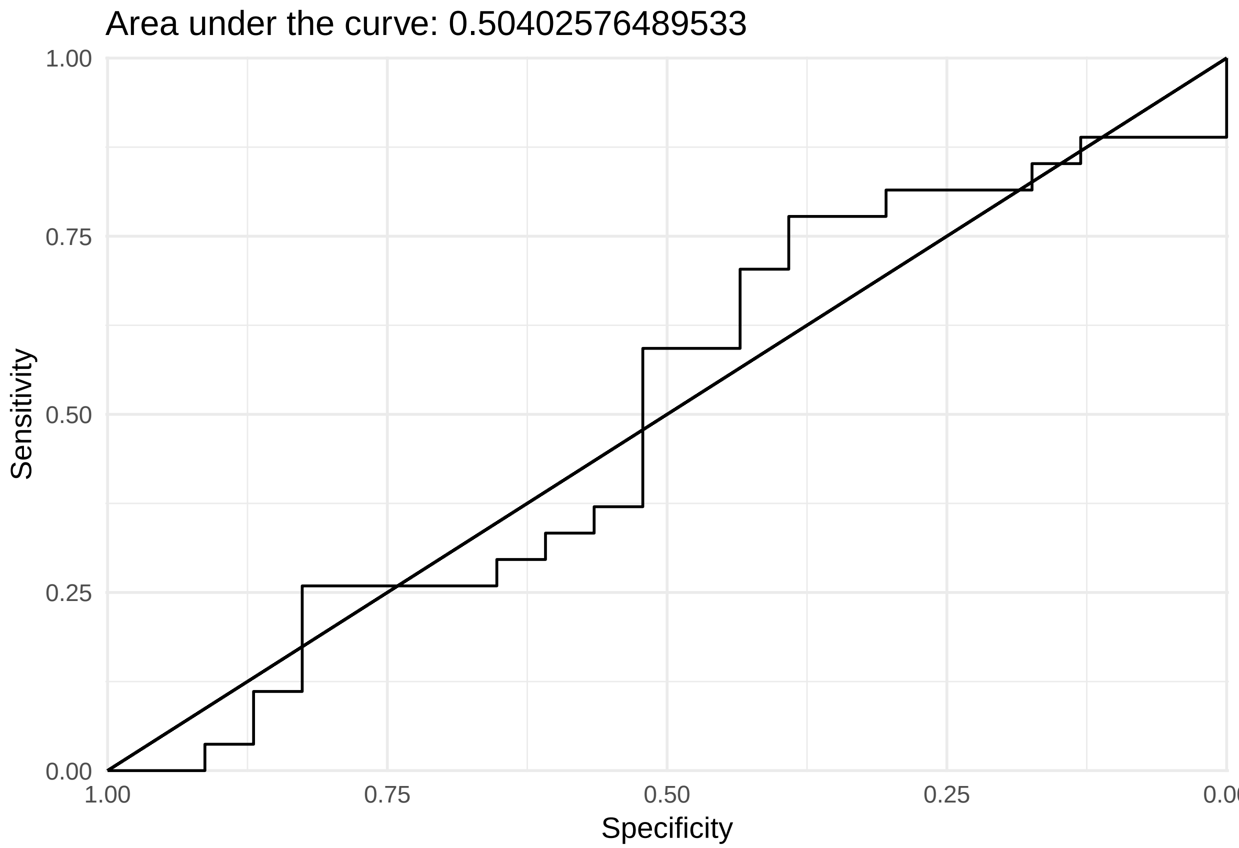
As with our regression model, we see a strong association of predictive power from the `REGION` variable.

```r
# Set n.trees to 10000
df_test$model_prediction_gbm_classifier <-
  predict(gradient_boosted_classifier, newdata = df_test %>%
            mutate(nrwpcent_class=as.character(nrwpcent_class)), n.trees = 10000, type = "response")

gbm_classifier_roc <- roc(df_test$nrwpcent_class, df_test$model_prediction_gbm_classifier)

auc_score <- gbm_classifier_roc$auc

cbind(rev(gbm_classifier_roc$specificities), rev(gbm_classifier_roc$sensitivities)) %>%
  as.data.frame() %>%
  rename('Specificity'=V1, 'Sensitivity'=V2) %>%
  ggplot(aes(x=Specificity, y=Sensitivity)) +
  geom_segment(aes(x = 0, y = 1, xend = 1,yend = 0), alpha = 0.5)  +
  geom_step() +
  scale_x_reverse(name = "Specificity",limits = c(1,0), expand = c(0.001,0.001)) +
  scale_y_continuous(name = "Sensitivity", limits = c(0,1), expand = c(0.001, 0.001)) +
  labs(title=paste("Area under the curve:", auc_score, sep=" ")) +
  theme_minimal()
```



The GBM classifier has an AUC in the test set at 0.504.

```
# Confusion matrix
predicted_probabilities_gbm <- df_test$model_prediction_gbm_classifier
predicted_probabilities_gbm[predicted_probabilities_gbm > 0.5] <- 1
predicted_probabilities_gbm[predicted_probabilities_gbm <= 0.5] <- 0
predicted_probabilities_gbm <- predicted_probabilities_gbm %>%
  as.vector %>% as.integer %>% as.factor
observed_gbm <- df_test$nrwpcent_class %>% as.factor
confusionMatrix(data=predicted_probabilities_gbm, reference=observed_gbm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 10 11
##          1 13 16
##
##                Accuracy : 0.52
##                  95% CI : (0.3742, 0.6634)
##     No Information Rate : 0.54
##     P-Value [Acc > NIR] : 0.6657
##
##                   Kappa : 0.0276
##  Mcnemar's Test P-Value : 0.8383
##
##             Sensitivity : 0.4348
```

```
##             Specificity : 0.5926
##          Pos Pred Value : 0.4762
##          Neg Pred Value : 0.5517
##              Prevalence : 0.4600
##          Detection Rate : 0.2000
##    Detection Prevalence : 0.4200
##       Balanced Accuracy : 0.5137
##
##        'Positive' Class : 0
##
```

Interestingly, the accuracy of this model is 52%, similar to the random forest model.

It should be noted that in a previous exercise using linear regression, we also had accuracy metrics in the range of ~50%.

Conclusions and Recommendations

In terms of the regression problem, we have the following test RMSE metrics:

Model	RMSE
Decision Tree	0.268
Random Forest	0.246
GBM	0.268

In terms of the classification problem, we have the following AUC and test accuracy metrics:

Model	AUC	Accuracy
Decision Tree	0.535	56%
Random Forest	0.550	52%
GBM	0.504	52%

In all tree-based models, we found `REGION` to be an important feature. If we wish to maximize test prediction performance for the regression task, we recommend deploying the random forest model since this had the lowest RMSE. For the classification problem, the pruned decision tree is recommended as it has the highest accuracy.