Bagged Trees and Random Forests

Applied Machine Learning for Educational Data Science

true

11/15/2021

Contents

[Updated: Sat, Nov 20, 2021 - 08:32:31]

The Concept of Bootstrap AGGregation (BAGGING)

The concept of bagging is based on the idea that predictions from an ensemble of models is better than the predictions from any single model. If we had a chance to randomly draw multiple samples from a population, and then to develop a prediction model for an outcome using each sample, the aggregated predictions from these multiple models would perform better due to the reduced model variance (aggregation would reduce noise due to sampling).

Due to the lack of access to population (even we assume there is a well defined population), we can mimic the sampling from a population by replacing it with **bootstrapping**. A **Bootstrap sample** is a random sample with replacement from the sample data.

The process of baggig is building separate models for each bootstrap sample and then apply all these models to a new observation for predicting the outcome. Finally, these predictions are aggregated in some form (e.g., taking average) to obtain a final prediction for the new observation. The idea of bagging can technically be applied any type of prediction model (e.g., KNNs, regression models). During the model process from each bootstrap sample, there is no regularization applied and models were developed to its full complexity. So, we obtain so many unbiased models. While each model has a large sample variance, we hope to reduce this sampling variance by aggregating the predictions from all these models at the end.

Bagged Trees for Predicting Readability Scores

Do It Yourself!

In this section, we will apply the idea of bagging to decision trees for predicting the readability scores. First, we import and prepare data for modeling. Then, we split the data into training and test pieces.

```
# Import the dataset

readability <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/ma

# Remove the variables with more than 80% missingness

require(finalfit)

missing_    <- ff_glimpse(readability)$Continuous
flag_na     <- which(as.numeric(missing_$missing_percent) > 80)
readability <- readability[,-flag_na]

# Write the recipe

require(recipes)

blueprint_readability <- recipe(x     = readability,
                                vars  = colnames(readability),
                                roles = c(rep('predictor',990),'outcome')) %>%
  step_zv(all_numeric()) %>%
  step_nzv(all_numeric()) %>%
  step_impute_mean(all_numeric()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_corr(all_numeric(),threshold=0.9)

# Train/Test Split

set.seed(10152021)  # for reproducibility

  loc      <- sample(1:nrow(readability), round(nrow(readability) * 0.9))
  read_tr  <- readability[loc, ]
  read_te  <- readability[-loc, ]
```

The code below will take a 1) bootstrap sample from training data, 2) develop a full tree model with no pruning, and 3) save the model object as an element of a list. We will repeat this process 10 times.

```
bag.models <- vector('list',10)

for(i in 1:10){

  # Bootstrap sample

    temp_rows <- sample(1:nrow(read_tr),nrow(read_tr),replace=TRUE)

    temp <- read_tr[temp_rows,]
  # Train the tree model with no pruning and no cross validation

    grid <- data.frame(cp=0)
    cv <- trainControl(method = "none")

    bag.models[[i]] <- caret::train(blueprint_readability,
                                    data      = temp,
                                    method    = 'rpart',
                                    tuneGrid  = grid,
                                    trControl = cv,
                                    control   = list(minsplit=20,
                                                     minbucket = 2,
                                                     maxdepth = 60))

}
```

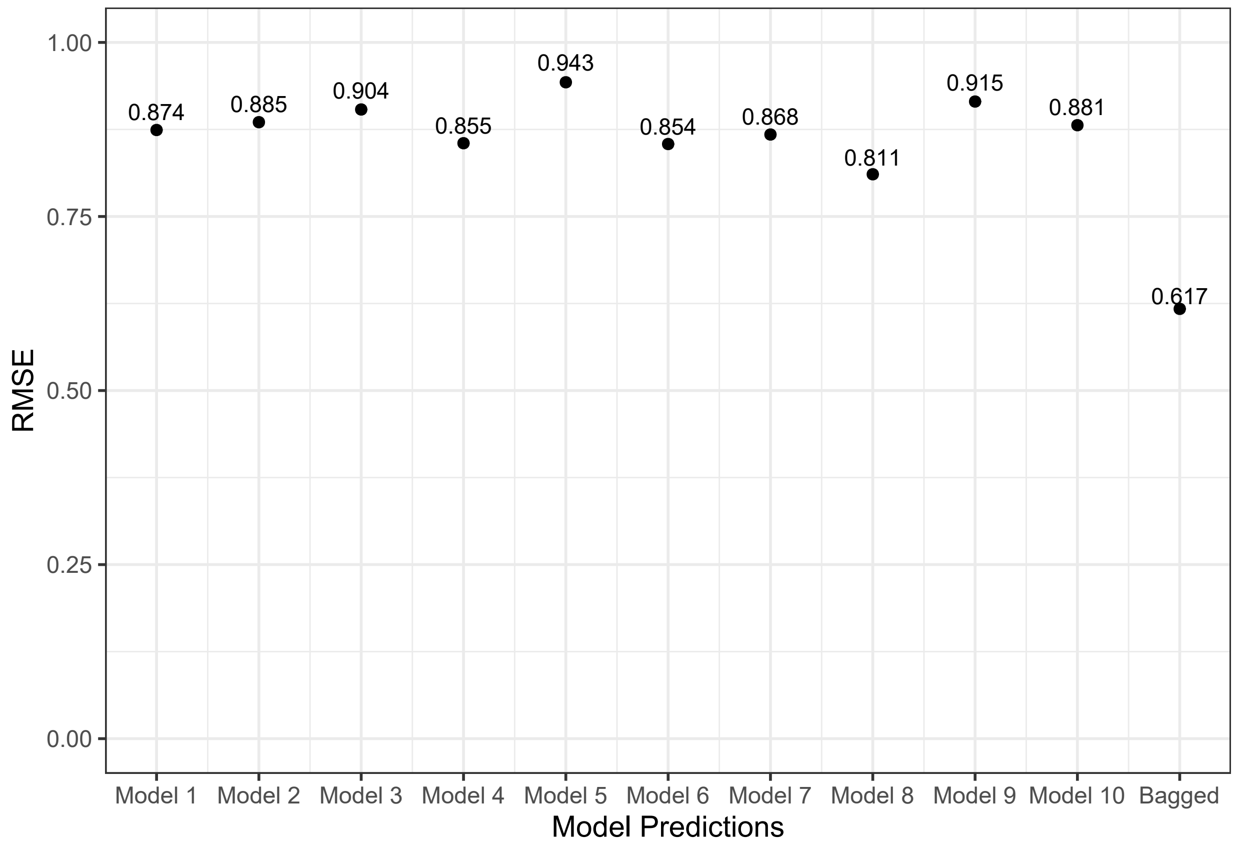
Now, we will use each of these models to predict the readability score for the test data. We will also average these predictions. Then, we will save the predictions in a matrix form to compare.

```
preds <- data.frame(obs = read_te[,c('target')])

preds$model1  <- predict(bag.models[[1]],read_te)
preds$model2  <- predict(bag.models[[2]],read_te)
preds$model3  <- predict(bag.models[[3]],read_te)
preds$model4  <- predict(bag.models[[4]],read_te)
preds$model5  <- predict(bag.models[[5]],read_te)
preds$model6  <- predict(bag.models[[6]],read_te)
preds$model7  <- predict(bag.models[[7]],read_te)
preds$model8  <- predict(bag.models[[8]],read_te)
preds$model9  <- predict(bag.models[[9]],read_te)
preds$model10 <- predict(bag.models[[10]],read_te)

preds$average <- rowMeans(preds[,2:11])

head(round(preds,3))
```

```
     obs model1 model2 model3 model4 model5 model6 model7 model8 model9 model10
1  0.246  0.588 -0.372  0.389  0.488  0.477 -0.435 -0.846 -0.203  0.015   0.173
2 -0.188 -1.524 -0.361 -0.548 -1.274 -0.387 -0.989 -0.438 -0.051  1.128  -0.370
3 -0.135  0.131 -0.421  0.561  0.452 -0.237  0.745 -0.152  0.365 -0.394   0.570
4  0.395  0.588  0.094  0.064  0.488  0.477  0.145  0.058  0.500  0.126  -0.739
5 -0.371 -1.021 -0.165 -1.211 -1.195  0.004 -1.816 -1.492 -0.595 -1.884  -0.812
6 -1.156 -1.262 -0.648 -0.004 -1.086 -0.295 -0.510 -1.000 -1.869 -1.148  -1.905
  average
1   0.027
2  -0.481
3   0.162
4   0.180
5  -1.019
6  -0.973
```

Now, let's compute the RMSE for the predicted scores from each model, and also the RMSE for average of predicted scores from all 10 tree models.

```r
p1 <- sqrt(mean((preds$obs - preds$model1)^2))
p2 <- sqrt(mean((preds$obs - preds$model2)^2))
p3 <- sqrt(mean((preds$obs - preds$model3)^2))
p4 <- sqrt(mean((preds$obs - preds$model4)^2))
p5 <- sqrt(mean((preds$obs - preds$model5)^2))
p6 <- sqrt(mean((preds$obs - preds$model6)^2))
p7 <- sqrt(mean((preds$obs - preds$model7)^2))
p8 <- sqrt(mean((preds$obs - preds$model8)^2))
p9 <- sqrt(mean((preds$obs - preds$model9)^2))
p10 <- sqrt(mean((preds$obs - preds$model10)^2))

p.ave <- sqrt(mean((preds$obs - preds$average)^2))


ggplot()+
  geom_point(aes(x = 1:11,y=c(p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p.ave)))+
  xlab('Model Predictions') +
  ylab('RMSE') +
  ylim(0,1) +
  scale_x_continuous(breaks = 1:11,
                     labels=c('Model 1','Model 2', 'Model 3', 'Model 4',
                              'Model 5','Model 6', 'Model 7', 'Model 8',
                              'Model 9','Model 10','Bagged'))+
  theme_bw()+
  annotate('text',
           x = 1:11,
           y=c(p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p.ave)*1.03,
           label = round(c(p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p.ave),3),
           cex=3)
```



As it is obvious, the bagging of 10 different tree models significantly improved the predictions on the test dataset.

Bagging via the `ranger` package and `caret::train()`

Instead of writing your own code to implement the idea of bagging for decision trees, we can use the `ranger` method via `caret::train()`.

```
require(caret)
require(ranger)

getModelInfo()$ranger$parameters
```

```
      parameter     class                         label
1          mtry   numeric #Randomly Selected Predictors
2     splitrule character                Splitting Rule
3 min.node.size   numeric            Minimal Node Size
```

```
?ranger
```

The `caret::train()` allow us manipulate three parameters while using the ranger method:

- *splitrule*: set this to 'variance' for regression problems with continuous outcome

- *min.node.size.*: this is identical to `minbucket` argument in the `rpart` method and indicates the minimum number of observations for each node.
- *mtry*: this is the most important parameter for this method and indicates the number of predictors to consider for developing tree models. For bagged decision trees, you can set this to the number of predictor variables in your model (we can change it for random forest models, coming later!)

```
# No cross validation

cv   <- trainControl(method = "none")

# Grid, running with all predictors available in the data (887)

grid <- expand.grid(mtry = 887,splitrule='variance',min.node.size=2)
grid
```

```
  mtry splitrule min.node.size
1  887  variance             2
```

```
# Bagging with 10 tree models

bagged.trees <- caret::train(blueprint_readability,
                             data      = read_tr,
                             method    = 'ranger',
                             trControl = cv,
                             tuneGrid  = grid,
                             num.trees = 10,
                             max.depth = 60)

bagged.trees$times
```

```
$everything
   user  system elapsed
  53.13    0.07   37.75

$final
   user  system elapsed
  33.27    0.02   17.85

$prediction
[1] NA NA NA
```

This took about 38 seconds to run. Let's check the predictions on the test dataset.

```
predicted_te <- predict(bagged.trees,read_te)

# RMSE

sqrt(mean((read_te$target - predicted_te)^2))
```

```
[1] 0.6556773
```

This is very similar to what we got in our DIY demonstration.

A couple things to note:

- When you set `max.depth=` argument within the `caret::train` function, it passes this to the `ranger` function. Try to set this number to as large as possible so you develop each tree to the full capacity.
- The penalty term is technically zero (`cp` parameter in the `rpart` function) while building each tree model. In Bagging, we deal with the model variance in a different way. Instead of applying a penalty term, we ensemble many unpenalized tree models to reduce the model variance.
- It is a little tricky to obtain reproducible results from this procedure. See this link to learn more about how to accomplish that.
- The number of trees (bootstrap samples) is a hyperparameter to tune. Conceptually, the model performance will improve as you increase the number of tree models used in Bagging; however, the performance will stabilize at some point. It is a tuning process to find the minimum number of tree models to use in Bagging to obtain the maximal model performance.

Tuning the Number of Tree Models in Bagging

Unfortunately, `caret::train` does not let us to define the `num.trees` argument as a hyperparameter in the grid search. So, the only way to search for the optimal number of trees is to use the `ranger` method via `caret::train` function and iterate over `num.trees` values from 1 to K (e.g., 200). Then, compare the model performance and pick the optimal number of tree models.

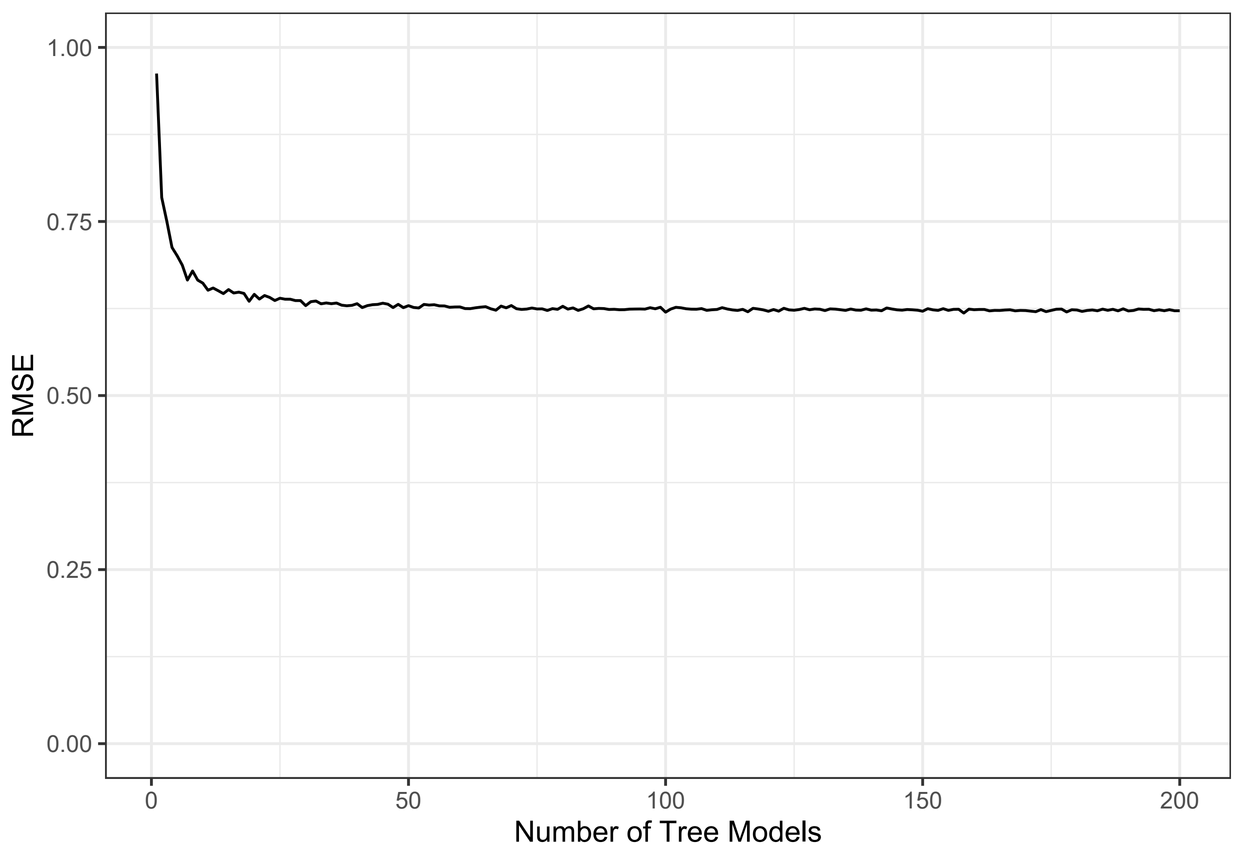
The code below implements this idea and saves the results from each iteration in a list object.

```
# Register multiple cores for parallel processing

    require(doParallel)

    ncores <- 10

    cl <- makePSOCKcluster(ncores)

    registerDoParallel(cl)

# Cross validation settings

    read_tr = read_tr[sample(nrow(read_tr)),]

    # Create 10 folds with equal size

    folds = cut(seq(1,nrow(read_tr)),breaks=10,labels=FALSE)

    # Create the list for each fold

    my.indices <- vector('list',10)
    for(i in 1:10){
      my.indices[[i]] <- which(folds!=i)
    }

    cv <- trainControl(method = "cv",
                       index  = my.indices)

# Grid Settings

    grid <- expand.grid(mtry = 887,splitrule='variance',min.node.size=2)
```

```r
# Run the bagged trees by iterating over num.trees values from 1 to 200

    bags <- vector('list',200)

    for(i in 1:200){

      bags[[i]] <- caret::train(blueprint_readability,
                                data      = read_tr,
                                method    = 'ranger',
                                trControl = cv,
                                tuneGrid  = grid,
                                num.trees = i,
                                max.depth = 60,)

    }

    # This can take a few hours to run.

stopCluster(cl)
```

Let's check the cross-validated RMSE for the bagged tree models with different number of trees.

```r
rmses <- c()

for(i in 1:200){

  rmses[i] = bags[[i]]$results$RMSE

}


ggplot()+
  geom_line(aes(x=1:200,y=rmses))+
  xlab('Number of Tree Models')+
  ylab('RMSE')+
  ylim(c(0,1))+
  theme_bw()
```



```
which.min(rmses)
```

```
[1] 158
```

This indicates that the RMSE stabilizes after roughly 50 tree models, and there is not much improvement. To be more precise, we can see that a bagged tree model with 158 trees gave the best result. Let's see how well this model performs on the test data.

```
# Predictions from a Bagged tree model with 158 trees

predicted_te <- predict(bags[[158]],read_te)

# MAE

mean(abs(read_te$target - predicted_te))
```

```
[1] 0.4816441
```

```
# RMSE

sqrt(mean((read_te$target - predicted_te)^2))
```

```
[1] 0.6040654
```

```
# R-square

cor(read_te$target,predicted_te)^2
```

```
[1] 0.6562741
```

Now, we can add this to our comparison list to remember how well this performs compared to other methods.

	R-square	MAE	RMSE
Linear Regression	0.644	0.522	0.644
Ridge Regression	0.727	0.435	0.536
Lasso Regression	0.725	0.434	0.538
KNN	0.623	0.500	0.629
Decision Tree	0.497	0.577	0.729
Bagged Trees	0.656	0.481	0.604

Random Forests for Predicting Readability Scores

Random Forests is an idea very similar to Bagging with an exception. In Random Forests, While we take a bootstrap sample of observations (random sample of rows in training data with replacement), we also take a random sample of columns for each split while developing a tree model. This allows us to develop tree models more independent of each other.

When there are certain important predictors related to the outcome, the tree models developed using all predictiors will be very similar to each other, particularly at the top, although we take bootstrap samples. These trees are going to be correlated to each which may reduce the efficiency in reducing the variance. By randomly sampling a certain number of predictors while developing each tree, we diversify the tree models. It turns out a diverse group of tree models do much better in predicting the outcome compared to a group of tree models very similar to each other.

We can use the same `ranger` package to fit the random forests models by only changing the `mtry` argument in our grid. Below, I set `mtry=300` indicating that it will randomly sample 300 predictors to consider for each split when developing each tree.

```
# No cross validation

cv   <- trainControl(method = "none")

# Grid, running with all predictors available in the data (887)

grid <- expand.grid(mtry = 300,splitrule='variance',min.node.size=2)
grid
```

```
  mtry splitrule min.node.size
1  300  variance             2
```

```
# Random Forest with 10 tree models

rforest <- caret::train(blueprint_readability,
                        data      = read_tr,
                        method    = 'ranger',
                        trControl = cv,
                        tuneGrid  = grid,
                        num.trees = 10,
                        max.depth = 60)

rforest$times
```

```
$everything
   user  system elapsed 
  43.88    0.35   38.91 

$final
   user  system elapsed 
  22.97    0.10   17.75 

$prediction
[1] NA NA NA
```

```
predicted_te <- predict(rforest,read_te)

# RMSE

sqrt(mean((read_te$target - predicted_te)^2))
```

```
[1] 0.638766
```

For random forests, there seems to be two hyperparameters to tune:

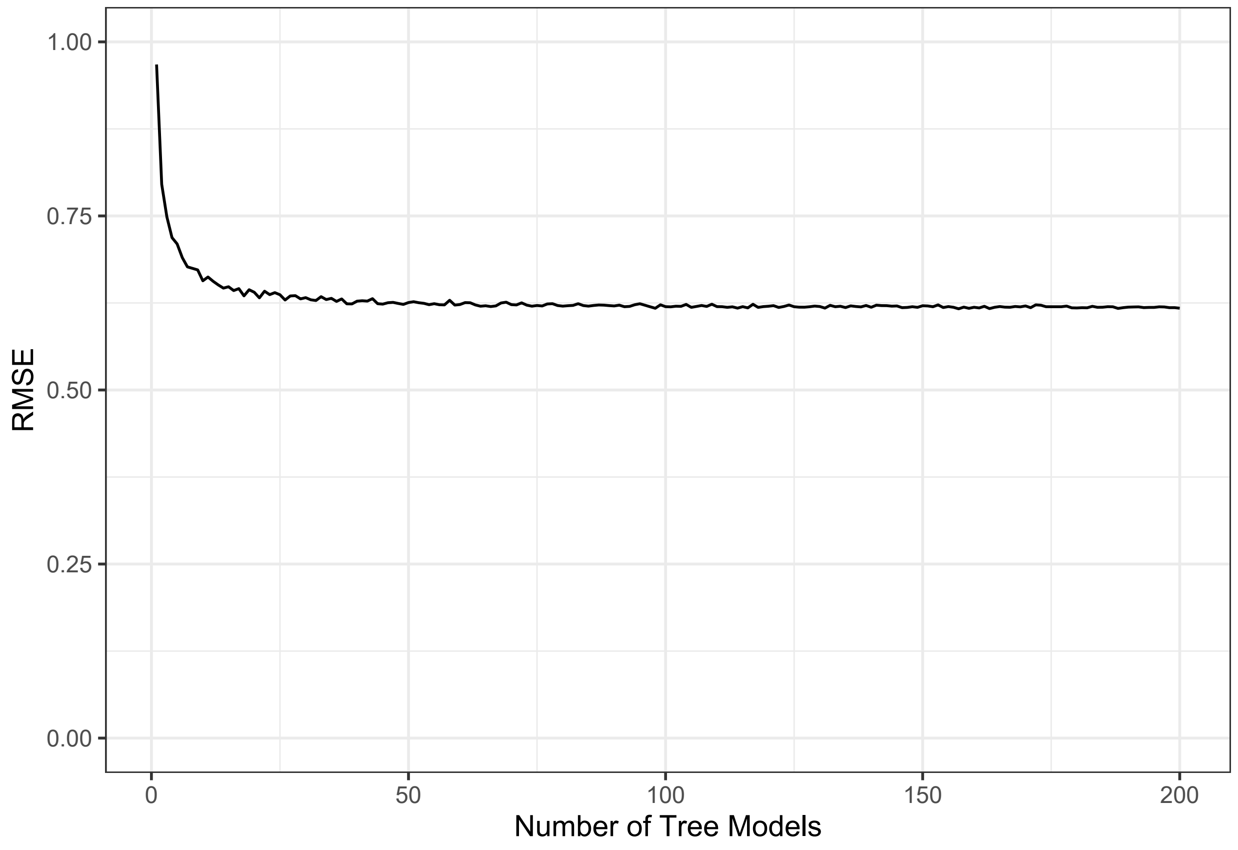
- `mtry`, the number of predictors to choose for each split during the tree model development
- `num.trees`, the number of trees.

As mentioned before, unfortunately, the `caret::train` only allows `mtry` in the grid search. For the number of trees, one should embed it in a `for` loop to iterate over for different number of trees. The code below hypothetically implements this idea by trying 10 different `mtry` values (100,150,200,250,300,350,400,450,500,550) and saves the results from each iteration in a list object. However, I haven't run it and it may take a very long time.

```
# Register multiple cores for parallel processing

    require(doParallel)

    ncores <- 10

    cl <- makePSOCKcluster(ncores)

    registerDoParallel(cl)

# Cross validation settings

    read_tr = read_tr[sample(nrow(read_tr)),]
    # Create 10 folds with equal size

    folds = cut(seq(1,nrow(read_tr)),breaks=10,labels=FALSE)

    # Create the list for each fold

    my.indices <- vector('list',10)
    for(i in 1:10){
      my.indices[[i]] <- which(folds!=i)
    }

    cv <- trainControl(method = "cv",
                       index  = my.indices)

# Grid Settings

    grid <- expand.grid(mtry = c(100,150,200,250,300,350,400,450,500,550),
                        splitrule='variance',
                        min.node.size=2)

# Run the bagged trees by iterating over num.trees values from 1 to 200

    bags <- vector('list',200)

    for(i in 1:200){

      bags[[i]] <- caret::train(blueprint_readability,
                                data      = read_tr,
                                method    = 'ranger',
                                trControl = cv,
                                tuneGrid  = grid,
                                num.trees = i,
                                max.depth = 60,)

    }

    # This can take a few hours to run.

    stopCluster(cl)
```

Instead, I run this only by fixing `mtry=300` and then iterate over the number of trees from 1 to 200 (as we did for bagged trees). Below are the results.

```
rmses <- c()

for(i in 1:200){

  rmses[i] = bags[[i]]$results$RMSE

}


ggplot()+
  geom_line(aes(x=1:200,y=rmses))+
  xlab('Number of Tree Models')+
  ylab('RMSE')+
  ylim(c(0,1))+
  theme_bw()
```



```
which.min(rmses)
```

```
[1] 157
```

This indicates that the RMSE stabilizes after roughly 50 tree models, and there is not much improvement. To be more precise, we can see that a bagged tree model with 158 trees gave the best result. Let's see how well this model performs on the test data.

```
# Predictions from a Bagged tree model with 158 trees

predicted_te <- predict(bags[[157]],read_te)

# MAE

mean(abs(read_te$target - predicted_te))
```

```
[1] 0.4715838
```

```
# RMSE

sqrt(mean((read_te$target - predicted_te)^2))
```

```
[1] 0.5967818
```

```
# R-square

cor(read_te$target,predicted_te)^2
```

```
[1] 0.6714306
```

Below is our comparison table with Random Forests added. As you see, there is a slight improvement over Bagged Trees. It is possible that we can improve this a little more by trying different values of `mtry` and find an optimal number.

	R-square	MAE	RMSE
Linear Regression	0.644	0.522	0.644
Ridge Regression	0.727	0.435	0.536
Lasso Regression	0.725	0.434	0.538
KNN	0.623	0.500	0.629
Decision Tree	0.497	0.577	0.729
Bagged Trees	0.656	0.481	0.604
Random Forests	0.671	0.471	0.596

Predicting Recidivism using Bagged Trees and Random Forests

Bagged Trees

Below is the code you can use to predict the probability of recidivism using bagged trees.

1. Import the recidivism dataset and initial preparation

```
# Import data

recidivism <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/mai

# Write the recipe


  # List of variable types

  outcome <- c('Recidivism_Arrest_Year2')

  categorical <- c('Residence_PUMA',
                   'Prison_Offense',
                   'Age_at_Release',
                   'Supervision_Level_First',
                   'Education_Level',
                   'Prison_Years',
```

```r
                   'Gender',
                   'Race',
                   'Gang_Affiliated',
                   'Prior_Arrest_Episodes_DVCharges',
                   'Prior_Arrest_Episodes_GunCharges',
                   'Prior_Conviction_Episodes_Viol',
                   'Prior_Conviction_Episodes_PPViolationCharges',
                   'Prior_Conviction_Episodes_DomesticViolenceCharges',
                   'Prior_Conviction_Episodes_GunCharges',
                   'Prior_Revocations_Parole',
                   'Prior_Revocations_Probation',
                   'Condition_MH_SA',
                   'Condition_Cog_Ed',
                   'Condition_Other',
                   'Violations_ElectronicMonitoring',
                   'Violations_Instruction',
                   'Violations_FailToReport',
                   'Violations_MoveWithoutPermission',
                   'Employment_Exempt')

numeric   <- c('Supervision_Risk_Score_First',
                 'Dependents',
                 'Prior_Arrest_Episodes_Felony',
                 'Prior_Arrest_Episodes_Misd',
                 'Prior_Arrest_Episodes_Violent',
                 'Prior_Arrest_Episodes_Property',
                 'Prior_Arrest_Episodes_Drug',
                 'Prior_Arrest_Episodes_PPViolationCharges',
                 'Prior_Conviction_Episodes_Felony',
                 'Prior_Conviction_Episodes_Misd',
                 'Prior_Conviction_Episodes_Prop',
                 'Prior_Conviction_Episodes_Drug',
                 'Delinquency_Reports',
                 'Program_Attendances',
                 'Program_UnexcusedAbsences',
                 'Residence_Changes',
                 'Avg_Days_per_DrugTest',
                 'Jobs_Per_Year')

props      <- c('DrugTests_THC_Positive',
                  'DrugTests_Cocaine_Positive',
                  'DrugTests_Meth_Positive',
                  'DrugTests_Other_Positive',
                  'Percent_Days_Employed')

# Convert all nominal, ordinal, and binary variables to factors
# Leave the rest as is

for(i in categorical){

  recidivism[,i] <- as.factor(recidivism[,i])

}
```

```
# For variables that represent proportions, add/substract a small number
# to 0s/1s for logit transformation

for(i in props){
  recidivism[,i] <- ifelse(recidivism[,i]==0,.0001,recidivism[,i])
  recidivism[,i] <- ifelse(recidivism[,i]==1,.9999,recidivism[,i])
}

# Blueprint for processing variables

require(recipes)

blueprint_recidivism <- recipe(x  = recidivism,
                               vars  = c(categorical,numeric,props,outcome),
                               roles = c(rep('predictor',48),'outcome')) %>%
  step_indicate_na(all_of(categorical),all_of(numeric),all_of(props)) %>%
  step_zv(all_numeric()) %>%
  step_impute_mean(all_of(numeric),all_of(props)) %>%
  step_impute_mode(all_of(categorical)) %>%
  step_logit(all_of(props)) %>%
  step_ns(all_of(numeric),all_of(props),deg_free=3) %>%
  step_normalize(paste0(numeric,'_ns_1'),
                 paste0(numeric,'_ns_2'),
                 paste0(numeric,'_ns_3'),
                 paste0(props,'_ns_1'),
                 paste0(props,'_ns_2'),
                 paste0(props,'_ns_3')) %>%
  step_dummy(all_of(categorical),one_hot=TRUE)  %>%
  step_num2factor(Recidivism_Arrest_Year2,
                transform = function(x) x + 1,
                levels=c('No','Yes'))
```

2. Train/Test Split

```
loc <- which(recidivism$Training_Sample==1)

recidivism_tr  <- recidivism[loc, ]
recidivism_te  <- recidivism[-loc, ]
```

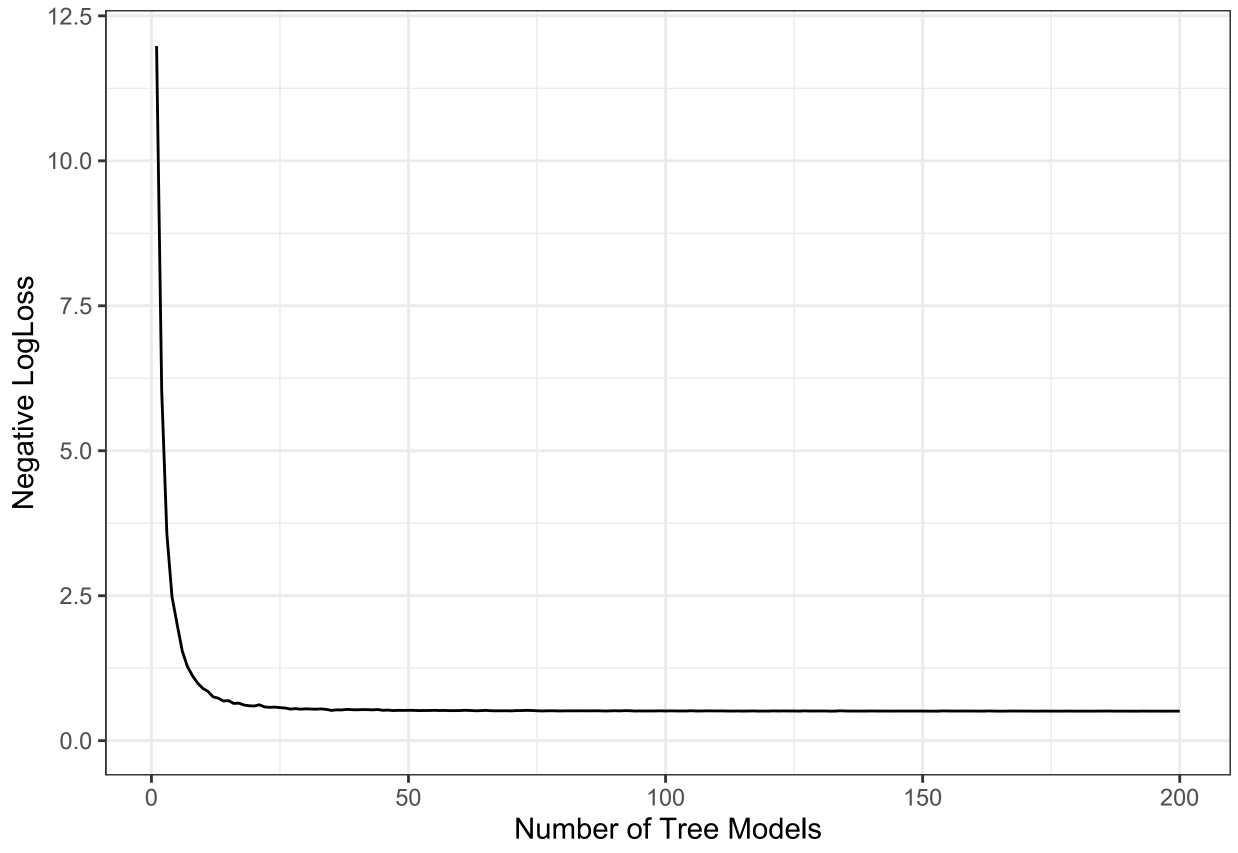
3. Tune the number of trees

```
# Cross validation settings

    set.seed(10302021) # for reproducibility

    recidivism_tr = recidivism_tr[sample(nrow(recidivism_tr)),]

  # Create 10 folds with equal size

    folds = cut(seq(1,nrow(recidivism_tr)),breaks=10,labels=FALSE)

  # Create the list for each fold
```

```r
    my.indices <- vector('list',10)
    for(i in 1:10){
      my.indices[[i]] <- which(folds!=i)
    }


  cv <- trainControl(method = "cv",
                     index  = my.indices,
                     classProbs = TRUE,
                     summaryFunction = mnLogLoss)

# Grid settings

grid <- expand.grid(mtry = 165,
                    splitrule='gini',
                    min.node.size=2)
grid

    # Grid, running with all predictors available in the data (165)
    # 165 is the number of predictors after we apply the recipe
    # to this dataset. We know this from earlier classes.
    # For bagged trees, we ask to use all of the predictors available in the dataset

    # Also, notice that I use 'gini' for splitrule because this is now a
    # classification problem

# Run the bagged trees by iterating over num.trees values from 1 to 200

    bags <- vector('list',200)

    for(i in 1:200){

      bags[[i]] <- caret::train(blueprint_recidivism,
                                data      = recidivism_tr,
                                method    = 'ranger',
                                trControl = cv,
                                tuneGrid  = grid,
                                metric    = 'logLoss',
                                num.trees = i,
                                max.depth = 60)
      print(i)
    }

    # This can take a few hours to run.
```

```r
logLoss_ <- c()

for(i in 1:200){

  logLoss_[i] = bags[[i]]$results$logLoss

}
```

```
ggplot()+
  geom_line(aes(x=1:200,y=logLoss_))+
  xlab('Number of Tree Models')+
  ylab('Negative LogLoss')+
  ylim(c(0,12))+
  theme_bw()
```



```
which.min(logLoss_)
```

```
[1] 197
```

4. Evaluate the model performance on the test set

```
# Predict the probabilities for the observations in the test dataset

predicted_te <- predict(bags[[197]], recidivism_te, type='prob')

dim(predicted_te)
```

```
[1] 5460    2
```

```r
head(predicted_te)
```

```
         No       Yes
1 0.9213198 0.0786802
2 0.6015228 0.3984772
3 0.5812183 0.4187817
4 0.5050761 0.4949239
5 0.8121827 0.1878173
6 0.6091371 0.3908629
```

```r
# Compute the AUC

require(cutpointr)

cut.obj <- cutpointr(x     = predicted_te$Yes,
                     class = recidivism_te$Recidivism_Arrest_Year2)

auc(cut.obj)
```

```
[1] 0.7224832
```

```r
# Confusion matrix assuming the threshold is 0.5

pred_class <- ifelse(predicted_te$Yes>.5,1,0)

confusion <- table(recidivism_te$Recidivism_Arrest_Year2,pred_class)

confusion
```

```
   pred_class
       0    1
  0 3967  179
  1 1143  171
```

```r
# True Negative Rate

confusion[1,1]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.9568259
```

```r
# False Positive Rate

confusion[1,2]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.04317414
```

```r
# True Positive Rate

confusion[2,2]/(confusion[2,1]+confusion[2,2])
```

```
[1] 0.130137
```

```r
# Precision

confusion[2,2]/(confusion[1,2]+confusion[2,2])
```

```
[1] 0.4885714
```

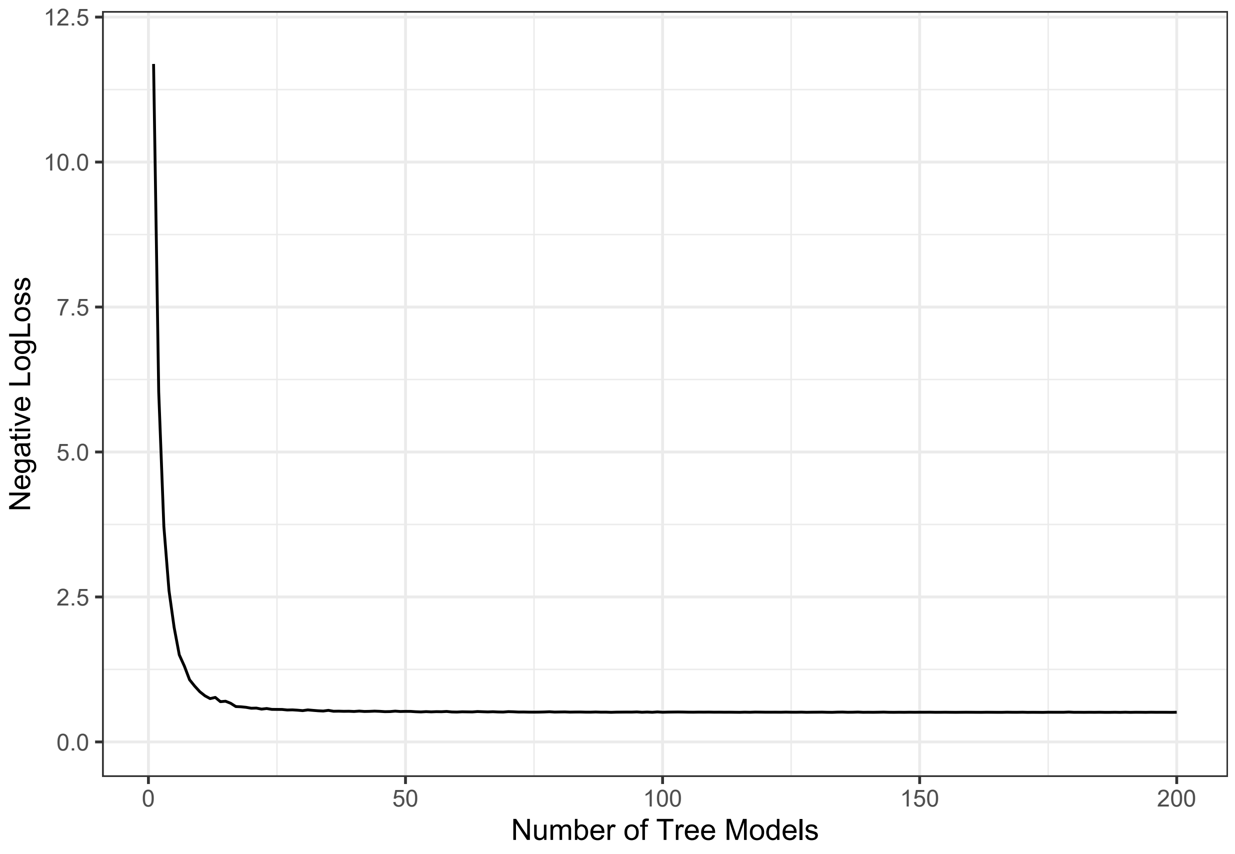
	-LL	AUC	ACC	TPR	TNR	FPR	PRE
Logistic Regression	0.5096	0.7192	0.755	0.142	0.949	0.051	0.471
Logistic Regression with Ridge Penalty	0.5111	0.7181	0.754	0.123	0.954	0.046	0.461
Logistic Regression with Lasso Penalty	0.5090	0.7200	0.754	0.127	0.952	0.048	0.458
Logistic Regression with Elastic Net	0.5091	0.7200	0.753	0.127	0.952	0.048	0.456
Decision Tree	0.5522	0.6521	0.752	0.089	0.962	0.038	0.427
Bagged Trees	0.5088	0.7225	0.758	0.130	0.957	0.043	0.489

Random Forests

Below is the code you can use to predict the probability of recidivism using bagged trees.

```r
# Grid settings

grid <- expand.grid(mtry = 80,
                    splitrule='gini',
                    min.node.size=2)
grid

    # The only difference for random forests is that I set mtry = 80

# Run the bagged trees by iterating over num.trees values from 1 to 200

    rfs <- vector('list',200)

    for(i in 1:200){

      rfs[[i]] <- caret::train(blueprint_recidivism,
                                data      = recidivism_tr,
                                method    = 'ranger',
                                trControl = cv,
                                tuneGrid  = grid,
                                metric    = 'logLoss',
                                num.trees = i,
                                max.depth = 60)
    }

    # This can take a few hours to run.
```

```r
logLoss_ <- c()

for(i in 1:200){

  logLoss_[i] = rfs[[i]]$results$logLoss
}


ggplot()+
  geom_line(aes(x=1:200,y=logLoss_))+
  xlab('Number of Tree Models')+
  ylab('Negative LogLoss')+
  ylim(c(0,12))+
  theme_bw()
```



```
which.min(logLoss_)
```

```
[1] 174
```

```
# Predict the probabilities for the observations in the test dataset

predicted_te <- predict(rfs[[174]], recidivism_te, type='prob')

# Compute the AUC

cut.obj <- cutpointr(x     = predicted_te$Yes,
                     class = recidivism_te$Recidivism_Arrest_Year2)

auc(cut.obj)
```

```
[1] 0.7241813
```

```r
# Confusion matrix assuming the threshold is 0.5

pred_class <- ifelse(predicted_te$Yes>.5,1,0)

confusion <- table(recidivism_te$Recidivism_Arrest_Year2,pred_class)

confusion
```

```
   pred_class
       0    1
  0 3993  153
  1 1149  165
```

```r
# True Negative Rate

confusion[1,1]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.963097
```

```r
# False Positive Rate

confusion[1,2]/(confusion[1,1]+confusion[1,2])
```

```
[1] 0.03690304
```

```r
# True Positive Rate

confusion[2,2]/(confusion[2,1]+confusion[2,2])
```

```
[1] 0.1255708
```

```r
# Precision

confusion[2,2]/(confusion[1,2]+confusion[2,2])
```

```
[1] 0.5188679
```

	-LL	AUC	ACC	TPR	TNR	FPR	PRE
Logistic Regression	0.5096	0.7192	0.755	0.142	0.949	0.051	0.471
Logistic Regression with Ridge Penalty	0.5111	0.7181	0.754	0.123	0.954	0.046	0.461
Logistic Regression with Lasso Penalty	0.5090	0.7200	0.754	0.127	0.952	0.048	0.458
Logistic Regression with Elastic Net	0.5091	0.7200	0.753	0.127	0.952	0.048	0.456
Decision Tree	0.5522	0.6521	0.752	0.089	0.962	0.038	0.427
Bagged Trees	0.5088	0.7225	0.758	0.130	0.957	0.043	0.489
Random Forests	0.5097	0.7242	0.762	0.126	0.963	0.037	0.519