The analysis and comparison of the tree-based method with different simulation assumption

Author: Moses Chen

Introduction

Nowadays, machine learning techniques are wildly used in many fields, which also include economics and finance. Therefore, it is necessary to combine our domain knowledge with machine learning skills. In this term paper, there are some powerful and useful algorithms such as decision tree, and pruned tree and some ensemble methods such as boosting and bagging would be introduced. Moreover, some analysis methods such as ROC curve and Confusion matrix would include as well. Additionally, this term paper can be separated into two main portions. In the first portion, each algorithm would be introduced one by one with a clear calculation explanation and their properties discussion. In the second portion, there is a simulation study that tries to partially replicate the result of (Utami, I.T., et al,2014). In the simulation process, there are two simulation ways with three different cases that would be provided.

A. The Introduction and practical calculation of algorithms

a. Decision Tree:

Compare with random forest, Adaboost and gradient boosting, the decision tree may not be the strongest algorithm to make predictions or classification. However, the decision tree has strong interpretability and it is the core concept behind all these competitive methods. Therefore, we will start from the fundamental part - The decision Tree method. The inducer of the decision tree that people used before was MC4(MLC++ C4.5), which is introduced by Kohavi et al (1997) and it is a top-down decision tree (TDDT) and now we used a slightly different version called CART. The main difference between CART and MC4 is that CART can support both numerical and categorical target variables. The following table is the steps of the Decision tree

table 1

Defined X_i(training vector) $\in R^n$ $for\ i = 1 \ldots . I, y(label\ vector) \in R^l\ and\ \theta = (j, t_m)$ means the candidate split with predictors j and threshold t at node m. The data at node m can be stated as Q_m with n_m obs	
Step1.	$Q_m^{left}(\theta) = \{(x, y)\ conditionally\ on\ x_j < t_m\}$ and $Q_m^{right}(\theta) = Q_m \setminus Q_m^{left}$
Step2.	Calculate the $G(Q_m, \theta) = \frac{n_m^{left}}{n_m} * H\left(Q_m^{left}(\theta)\right) + \frac{n_m^{right}}{n_m} * H\left(Q_m^{right}(\theta)\right)$ (H means loss function, which will be introduced later on.)

Step3. Select the parameter $\theta^* = argmin\ G(Q_m, \theta)$

Step4. Recursive the $Q_m^{left}(\theta)$ and $Q_m^{right}(\theta)$ until the obs in terminal nodes = 1 or smaller than the number of terminal nodes threshold and find the result of prediction.

i. The detail of loss function criteria of the Decision Tree

Now, we go to the details of the criterion to divide the prediction space. The ways to divide prediction spaces differ between regression and classification trees. For the regression tree, there are two ways, one is called Mean squared error (for continuous target variable) and another is called Poisson deviance (for frequency target variable)

$H(Q_m) = \frac{1}{n_m} * \sum_{y \in Q_m} (y_i - \overline{y_m})^2 \quad for\ i = 1,2, \ldots . N$, for RSS

$H(Q_m) = \frac{1}{n_m} \sum_{y \in Q_m} (y \log \frac{y}{\bar{y}_m} - y + \bar{y}_m) \ for\ i = 1,2, .. N$ for Poisson Deviance.

Therefore, we need to use these criteria to experiment with different predictors with a different thresholds to reach our goal. For the classification tree, there are two main loss functions that people usually used as criteria to divide the prediction space, which is the Gini index and Entropy.

Gini index = $\sum_k p_{mk} * (1 - p_{mk}) \ for\ p_{mk} = \frac{1}{n_m} \sum_{y \epsilon Q_m} I(y = k)$

Entropy = $-\sum_k p_{mk} * \log(p_{mk}) \ for\ p_{mk} = \frac{1}{n_m} \sum_{y \epsilon Q_m} I(y = k)$

p_{mk} means the proportion of dependent variable in node m from each k class. To provide a clear explanation, I set up a very simple dataset. The dependent variable here is whether individuals will buy the lollipop or not. The independent variables are gender and age. My assumption here is that as long as they are younger than 10 years old, regardless of their gender, they will consume lollipops.

Table 2

Buy or Not	Gender	Age
yes	boy	6
no	girl	18

yes	girl	7
no	boy	20

Before dividing any prediction space, we can calculate the Gini index at first, in this case, we only have 1 region. ½*(1-½) + ½*(1-½) = ½. 1/2 (two yes two no). It shows that it is quite impure. If we used gender as a separator, we will get the exactly same result.
½ [½*(1-½)+½*(1-½)] + ½ [½*(1-½)+½*(1-½)] = ½.
In contrast, we take Age as a separator and assume that if age is smaller than 10 we classify it as yes, otherwise no. ½[1*(1-1)+0*(1-0)]+½[0*(1-0)+1*(1-1)] = 0. Then we found that we make a great progression from ½ to 0.

The outcome is similar for entropy.
Original dataset: -½*log(½) -½*log(½) = 1 (here I assume the base of log is 2 for easier calculation reasons, it does not matter if it is 2 or 10)
Taking gender as separator: ½[-½*log(½) -½*log(½)] + ½[-½*log(½) -½*log(½)] = 1 In other words, we did not have any progress from the original case.

Taking age as a separator, on the left-hand region is all the dependent variable with yes and the left is all the dependent variable with no: ½[-1*log(1) - 0] + ½[0 -1*log(1)] = tends to 0
Because log(0) does not exist, I assume here that 0 is tends to 0 and 1 is tends to 1.
Therefore, we can easily observe the concept that if the proportion of "yes" closed to 0.5 entropy and Gini index would reach the largest point, and conversely, it would reach the lowest point at the proportion = 0 or 1.

ii. Pruning the decision tree:

The reason why we need to prune the decision tree is that we want to avoid overfitting, which will occur if we take into account the error term or noise when we establish the decision tree model. Consequently, it may show very good performance on the training dataset but bad performance on the test dataset.
The general function of pruning is shown as follows.

$$R_\alpha = R(T) + \alpha * |T| \; and \; R(T) = RSS$$

A larger alpha means more strict penalization because our goal is to make R_α as small as possible. In this case, our tree will be relatively small. In contrast, if alpha = 0, then our tree would be exactly the same as the tree without pruning. To be more specific, several parameters can be set up for pruning the tree in the R function.

Table 3

Parameter	Meaning
min split	The threshold of the minimum number of individuals in this node to let this node split further. (Default is 20)
min bucket	The minimum number of individuals in the terminal node.
max depth	The maximum layer of the decision tree
cp	Explanation is below

cp formula is transformed from R_α and $R_{cp} = R(T) + cp * |T| * R(T_0)$

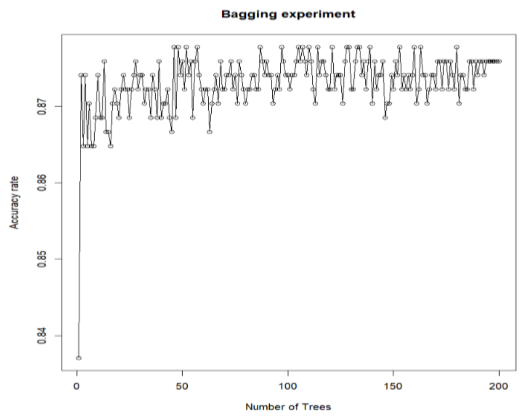
$R(T_0)$ means the RSS when there is no split. cp here is easier to control compared with alpha because it is unitless. If cp = 1, then we will not have any split in this tree because we cannot make any improvement for R_{cp}, even if we find the best dividing way which makes R(T) = 0, our R_{cp} will still equal to $R(T_0)$. In contrast, if cp = 0, then we can grow the tree as big as we want. In other words, cp is like a threshold to control the size of our tree. The simulation study will be shown in the next chapter, now we go to the random forest portion first.

b. Bagging and Random Forests

i. Bagging

Before talking about the random forest, we need to know about the bootstrapping or bagging method, because the concept of random forest is the extension of bagging. The reason why we need bagging is that we want to avoid high variance from only one tree. Therefore, we take the subset of the sample. Usually, it is ⅔ of a sample. Then we grow a full (without tuning) tree by the RSS, Poisson Deviance, Gini index, or Entropy like we have done in the decision tree. Next, we take ⅔ of the sample again and so on and so forth, and during the process, some individuals will duplicate in subsample but it does not matter in bagging. Finally, we will have many trees and we use the voting method for the classification problem and we take the mean for the regression problem. To be more specific, for instance, if we have a total of eight trees and 6 trees vote for yes and 2 trees vote for no for the classification problem, then we will predict as yes. Besides, a higher number of trees will not increase biased but maybe decrease the variance. However, calculating a high number of trees can be very time-consuming. Breiman. L (2001) mentioned that the accuracy rate of the infinity number of trees should converge. Therefore, it seems unnecessary to use a very large number of trees and we can have a similar outcome. I also try to do the same experiment. Figure2 shows a accuracy rate growing up and converging trend.

figure 1



For a more clear explanation, the following table is provided.

table 4

Defined X_i(training vector) $\in R^n$ $for\ i = 1 \ldots. I, y(label\ vector) \in R^l\ and\ X\ and\ y\ are\ from\ dataset\ S$ and inducer "I" is the process in table 1	
Step1.	For i = 1 to T {
Step2.	S' = bootstrap sample from S (i.i.d. sample with replacement)
Step3.	C_i(x) = I(S')}
Step4.	$C^*(x) = \arg\max \sum_{i\ \epsilon\ C_i(x)=y} 1$ (Voting for the final result)

ii. Random Forest

For random forest, it is quite similar to bagging, but the difference is it will not takes into account all the predictors. Typically, it only considers part of predictors as a candidate for splitting the tree. It is because if we consider all the predictors, maybe there are some predictors with a much stronger ability to make predictions than others. Consequently, these predictors are applied in almost every tree we establish, and therefore these trees are highly correlated. Finally, it may still have high variance because of a strong correlation between trees. To get rid of this issue, random forests randomly choose only part of the predictors to constrain the tree to be uncorrelated with other trees.

c. Boosting method and Adaboost

i. Boosting

The main difference between boosting and bagging is each draw for the bagging method is independent and identically distributed and conversely, each draw for boosting method is dependent. Moreover, for boosting method, the weight of individuals is different and the sub-sample we obtain mainly depends on the error of the previous tree. In other words, some individuals may have a higher opportunity to be drawn than other individuals. Because the subsample is different and the misclassified individuals will be cared for more, so each tree is like compensation for the previous tree. Normally, the size of a tree in boosting method is quite small, even just a stump which is only one split and two leaves (Iba & Langley, 1992). Therefore, the accuracy for each tree is very weak but if we have many small trees then the accuracy can improve step by step. For each step, we can also control the shrinkage parameter **λ** to force the model to improve slower. The **λ** will be set as between 0.01 to 0.001 usually. Besides, we also need to be careful that too many trees for boosting method may cause overfitting, which is different from bagging.

ii. Adaboost

There are some boosting methods that can be combined with a decision tree, which include Adaboost, gradient boosting, and XGboost. In this paragraph, Adaboost will be introduced. Adaboost is the abbreviation of adaptive boosting, it will consider the error of the previous tree and makes compensation in the following tree. In the beginning, the weight of each tree is the same. For instance, if we have 8 rows, then the weight of each row is ⅛. Then we used either RSS, Gini index, Entropy, etc to create the first stump (all the trees in Adaboost is stumps). After that, there are two things we need to care about. 1. How important this tree is (it depends on how good the prediction outcome). 2. How to change the weight of each observation (it depends on which observation is misclassified). The misclassified individuals are likely to show up more than once in the following subsample.

From the math perspective, let’s give a function to define how important this tree is.

$$\text{alpha} = \frac{1}{2} * \ln\left[\frac{1 - Total\ error}{\text{Total error}}\right]$$

When plotting it in the R studio, it would be looks like the following graph.

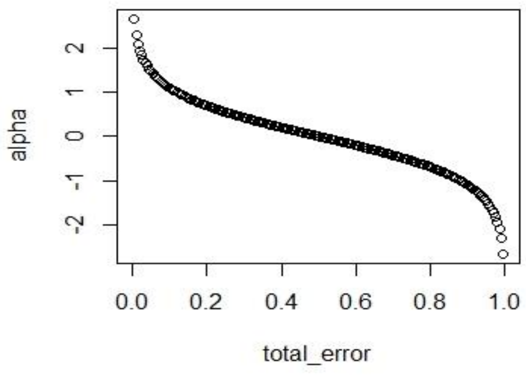


Figure 2

```
total_error <- seq(0,1,by=0.005)
alpha <- c()
alpha_cal <- function(total_error){
  alpha <- 1/2*log((1-total_error)/total_error)
}
alpha <- alpha_cal(total_error)
plot(total_error,alpha)
```

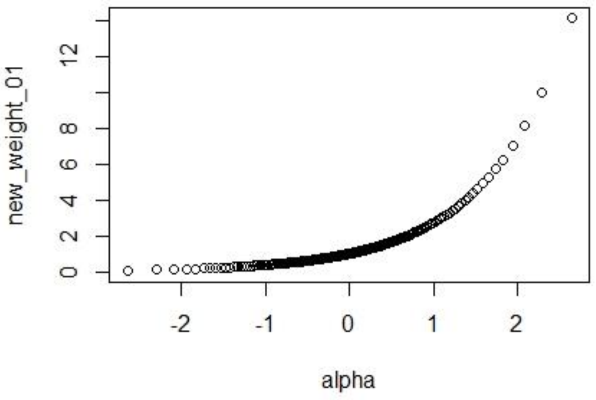
As we can observe, if the error rate is closed to 0.5, it means that the capability of predicting this tree is similar to guessing so alpha tends to 0. If the error rate is closed to 1, it means that this tree tends to predict in the exact opposite way so the alpha tends to minus infinity. On the other hand, if it almost predicts everything correct, then the alpha tends to infinity. During the final voting process, this tree will be very important if it has a high alpha value.

For the weight adjustment formula, there are two cases, one is for correctly classified individuals and one is for the misclassified individual.

1. New weight = old weight $* \ e^{\alpha} \ (for\ misclassified\ individual)$
2. New weight = old weight $* \ e^{-\alpha} \ (for\ correctly\ classified\ individual)$

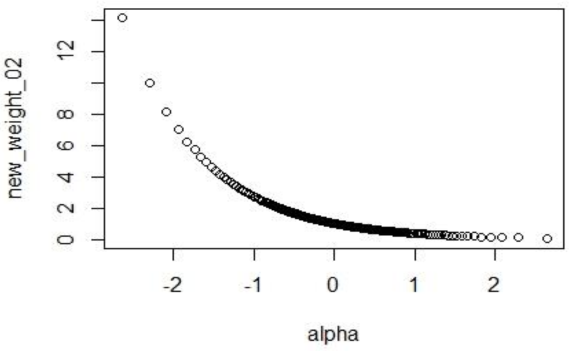
The plot of it will be shown on the next page.

Figure 3



As we can observe from Figure 3, if an important tree (large alpha) misclassified an individual, then the weight of that individual would increase a lot. In contrast, if a tree makes many mistakes, then the weight of misclassified individuals from that tree will almost not change.

Figure 4



As we can observe from Figure 4, if an important tree correctly classified the individual, then the correctly classified individual would barely change its weight. In contrast, if a tree makes many mistakes, but it makes a correct classification for this individual, then that individual will increase its weight significantly. The following table provides a short conclusion.

table 5

Assuming training set = S, with n = number of observations, inducer "I" = table 1 process but only a stump is allowed, T = number of trials.	
Step1.	Taking S = S' and assign weight as 1/n for each observation.
Step2.	For i = 1 to T{

Step3. $C_i = I(S')$

Step4. $\alpha = \frac{1}{2} * \ln\left[\frac{1-Total\ error}{\text{Total error}}\right]$

Step5. For each $x_j \epsilon S', if\ C_i(X_j) = y_i\ then\ weight =$ old weight $* e^{-\alpha}$ $, otherwise, weight =$ old weight $* e^{\alpha}$

Step6. Normalize to the weights of instances so the total weight of S' is 1}

Step7. $\text{C}^*(x) = arg \max_{y \in Y} \sum_{i \in C_i(x) = y} \alpha_i$

To be more specific, let's expand the dataset from Table 2 for calculation.

Table 6

Buy or Not	Gender	Age	weight
yes	boy	6	1/5
no	girl	15	1/5
yes	girl	7	1/5
no	girl	20	1/5
yes	girl	15	1/5

By the Gini index, age is a better separator than gender. Therefore, in the first stump, we use age < 15 as the separator and the error is ⅕. Therefore, the alpha = 1/2* ln(4) = 0.693. After that, we calculate the new weight. Misclassified individual weight = ⅕ * e^{α} = 0.4 Correctly classified individual weight = ⅕ * $e^{-\alpha}$ = 0.1. Therefore, the updated dataset will be like the table following.

Table 7

Buy or Not	Gender	Age	weight	Standardize weight
yes	boy	6	0.1	0.125
no	girl	15	0.1	0.125
yes	girl	7	0.1	0.125
no	girl	20	0.1	0.125
yes	girl	15	0.4	0.5

Next, I would randomly choose five numbers from 0 to 1 to decide the composition of the subsample. For instance, I choose 0.1, 0.3, 0.6, 0.9,1, then we can find the corresponding individual from the sum of standardized weight. Like following Table 8. Row five in Table 4 is the misclassified individual. So, for the sake of compensation, it shows up three times in the subsample (Table 5). Following, we calculate the Gini index again to determine the separator and again find the alpha and weight. By this iteration, we can create many stumps and at the end, every stump will make a prediction. The prediction result can only be yes or no, so we have two groups in this case. Next, we calculate the summation of alpha in these two groups to see which group is more important, then we take the result from that group as our final prediction.

Table 8

Buy or Not	Gender	Age	weight
yes	boy	6	1/5
yes	girl	7	1/5
yes	girl	15	1/5
yes	girl	15	1/5
yes	girl	15	1/5

In general, because the boosting method would boost the model according to the misclassified observations, so the boosting method would be very sensitive to outliers or noise (because usually outliers are misclassified). In contrast, Breiman. L (2001) states that random forest is relatively robust to noise or outliers, so compare with random forest, Adaboost has a higher requirement with the dataset. Khoshgoftaar, T.M. (2011) also mentions that the bagging method is generally better than boosting in the environment with noise. Besides, Adaboost needs to reweight each observation at each iteration, so usually, it is more time-consuming than random forest (Breiman. L, 2001).

B. Simulation Study

The following portion will partially replicate the result from (Utami, I.T., et al, 2014). Utami, I.T., et al (2014) use three kinds of data generating processes to test the capability of a decision tree, bagging method, support vector machine(SVM) with linear kernel, SVM with the polynomial kernel, and SVM with Radical kernel. However, to deepen the study of the tree-based method. SVM will not be discussed in this term paper but the bagging boosting method and a pruned tree will be discussed instead. Here, I will use a random forest to represent the bagging method, and 50 trees and 500 trees will be tested. For pruning a tree, because it is very time-consuming, I will only prune the tree by the complexity parameter (CP) value. For boosting method, Adaboost will be discussed and there are 100 stumps will be used. Besides, after using DGP, Utami, I.T., et al (2014) create 1200 individuals and used 70% of it as training data and 30% of the dataset as test data. Next, Utami, I.T., et al (2014) totally simulate it 5000 times and calculate the mean of misclassification and standard error for each algorithm. But here I will partially modify the rule. I will simulate once but with 50000 total observations and simulate 1000 times with only 1200 individuals like what the authors did in the paper. Finally, compare the result with the same case but different simulation way and with the result between different cases.

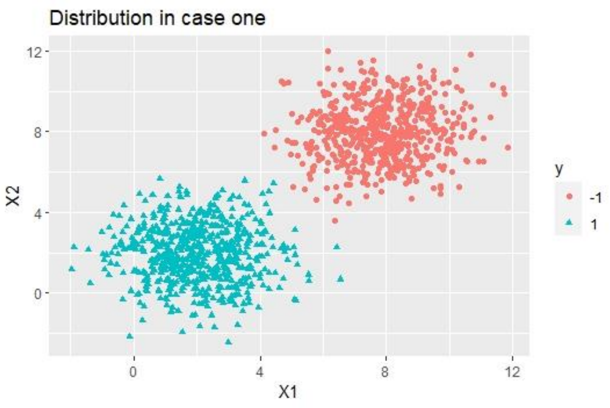
a. The set-up of three data generation process

In (Utami, I.T., et al, 2014), the first data generating process (case 1) follows the multivariate normal distribution with

$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \ \mu_2 = \begin{bmatrix} 8 \\ 8 \end{bmatrix} \text{ and } \text{sigma}_1 = \text{sigma}_2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} .$$

It means that we only have two classes, which are 1 and -1 and we have two explanatory variables. μ_1 is the mean for class 1 and μ_2 is the class for -1 and they have the same covariance matrix. Each class has 600 individuals, therefore we have a total of 1200 individuals in the dataset. In case one, Utami, I.T. et al, (2014) assume that the classes are perfectly linear separable.

figure 5



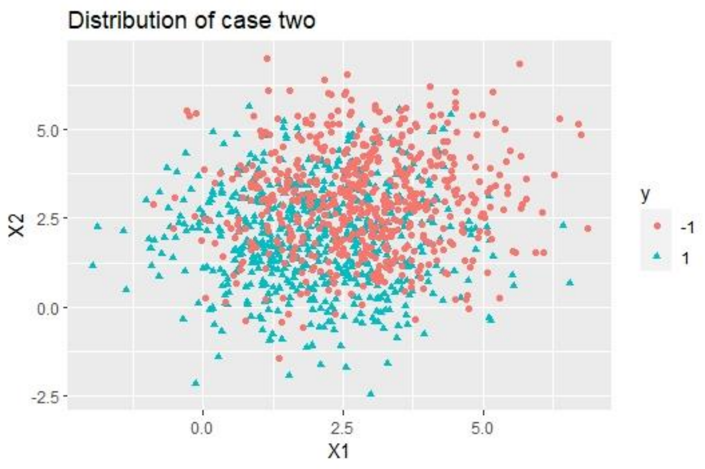
As we can observe from figure 5, the case1 dataset can be separated by a straight line, and the data generating process coding is shown as follows.

```r
dgp_01 <- function(mean1,mean2){
  sample_size <- 600
  sample_meanvector <- c(mean1, mean1)
  sample_covariance_matrix <- matrix(c(2, 0, 0, 2),
                                     ncol = 2)
  sample_distribution <- mvrnorm(n = sample_size,
                                 mu = sample_meanvector,
                                 Sigma = sample_covariance_matrix)
  positive <- data.frame(sample_distribution)
  positive['y'] <- rep(1,600)
  sample_meanvector <- c(mean2, mean2)#just need to change the mean, all the
 other are the same
  sample_minus <- mvrnorm(n = sample_size,
                          mu = sample_meanvector,
                          Sigma = sample_covariance_matrix)
  negative <- data.frame(sample_minus)
  negative['y'] <- rep(-1,600)
  data <- rbind(positive,negative)
}
```

For case 2, the parameter of multivariate normal distribution becomes

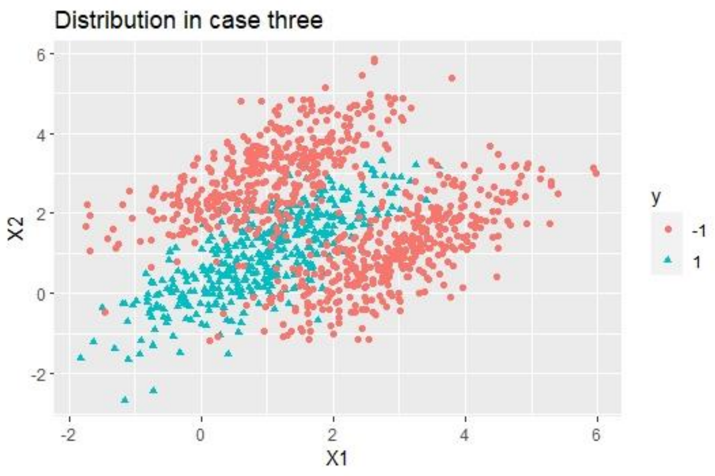
$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ $\mu_2 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$ and $\text{sigma}_1 = \text{sigma}_2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$. There are still 600 individuals for +1 and 600 individuals for -1. In this case, the individuals can be linear separable but not perfectly separated. figure 6 shows its distribution.

figure 6



We can use the same DGP from case1 to case2, but we need to change the μ_2 $to \begin{bmatrix} 3 \\ 3 \end{bmatrix}$. For case 3, we have $\mu_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ $\mu_2 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$ $\mu_3 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$ and $\text{sigma}_1 = \text{sigma}_2 = \text{sigma}_3 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}$. In this case, μ_1 is the mean for class +1 and μ_2 and μ_3 are the means for the -1 class. Each of μ_1, $\mu_{2,}$and μ_3 has 400 individuals so we have 1200 individuals in total as well. The distribution of case 3 is shown in figure 7.

figure 7



As we can see, in case 3, the observations cannot be classified correctly via a linear model. The data generating process code is shown as follows.

```r
dgp_03 <- function(N){
  sample_size <- N
  sample_meanvector <- c(1, 1)
  sample_covariance_matrix <- matrix(c(1, 0.7, 0.7, 1),
                                     ncol = 2)
  positive_distribution <- mvrnorm(n = sample_size,
                                   mu = sample_meanvector,
                                   Sigma = sample_covariance_matrix)
  positive <- data.frame(positive_distribution)
  positive['y'] <- rep(1,N)
  sample_meanvector <- c(1, 3)
  sample_minus_01 <- mvrnorm(n = sample_size,
                             mu = sample_meanvector,
                             Sigma = sample_covariance_matrix)
  negative <- data.frame(sample_minus_01)
  negative['y'] <- rep(-1,N)
  sample_meanvector <- c(3, 1)
  sample_minus_02 <- mvrnorm(n = sample_size,
                             mu = sample_meanvector,
                             Sigma = sample_covariance_matrix)
  negative_02 <- data.frame(sample_minus_02)
  negative_02['y'] <- rep(-1,N)
  data <- rbind(positive,negative,negative_02)
}
```

b. The simulation result for the case1(perfect linearly separable)

figure 8

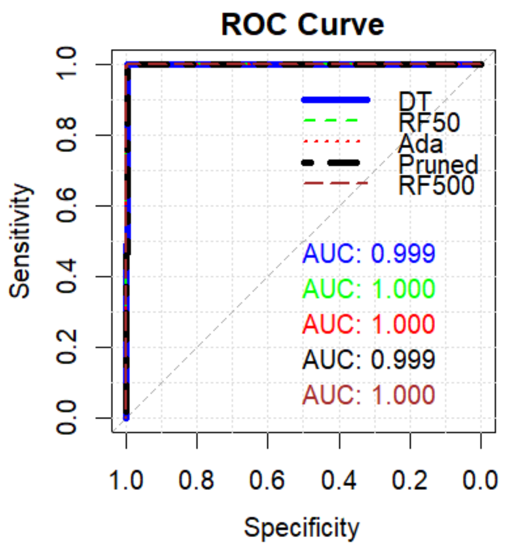
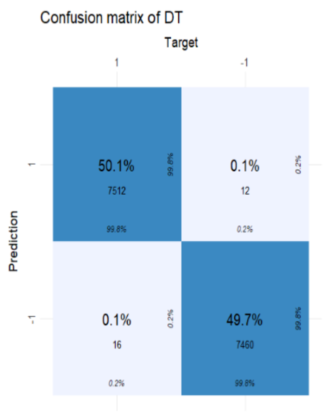


figure 8 shows the result of 50000 observations with once simulation. Here sensitivity means "true positive rate" and specificity means "true negative rate". In order to understand the logic behind ROC curve, we need to know the concept behind classification algorithm is that we would obtain two probability as prediction outcome such as p and 1-p. Assuming p is for positive cases and 1-p is for negative cases. Now we need to define a threshold to determine our result. The threshold can differ from different fields. For some specific field, the threshold needs to be very strict, such as medical test. For instance, even the probability for getting a cancer is 0.3, but that patient may still be classified as positive. Going back to the ROC curve explanation, we can start from the bottom left point, where specificity = 1. It means that the threshold needs to let all the negative cases be correctly classified. In that scenario, it would be very hard to correctly classified all positive cases because some of them may be classified as negative. After that, it would go to the upper-right direction. In this process, the threshold would be eased and finally, there is no need to classified any negative case correctly (specificity = 0), so all the positive cases would definitely be classified correctly. From figure 8 we can find that almost all algorithms can classify all the individuals correctly even in the very strict threshold. Moreover, AUC means “area under the curve”. If AUC =1 means that exist at least one threshold can correctly classify all the positive and negative cases. If AUC = 0.5, it means the prediction outcome is similar to guessing. If AUC lower than 0.5, it means the prediction is on the opposite way. Despite of the fact that DT (decision tree) and pruned tree have a little bit lower number of AUC, but they still perform very well.



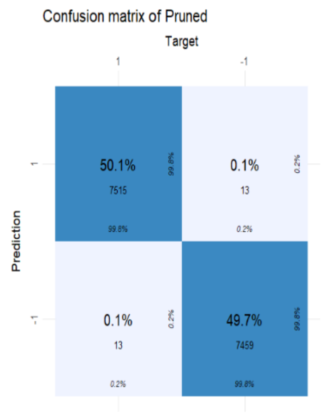
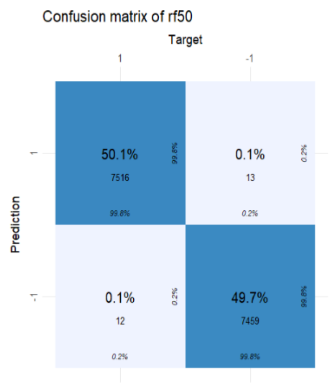


figure 9 and figure 10



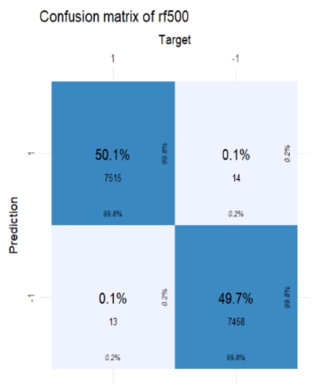


figure 11, figure 12

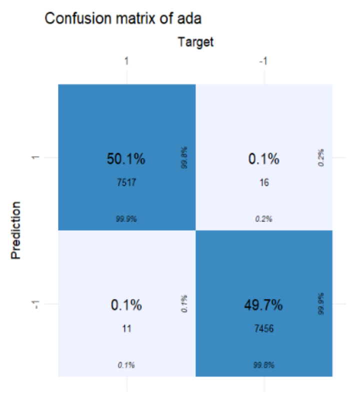


figure 13

To be more specific, from figure 9 and figure 10, we know that decision trees and pruned trees only have 28 and 26 misclassified cases from 50000 individuals. From figure 11, figure 12 and figure 13, it shows that only 25, 27, and 27 misclassified cases for rf50, rf500, and Ada respectively. In conclusion, all the algorithms perform very well in case one once simulation.

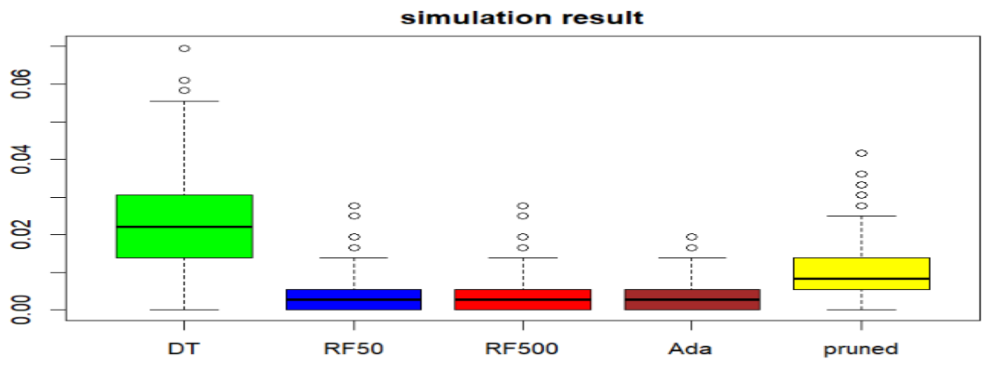
From the 1000 times simulation, we can get the following table and figure.

table 9

Percentage of average and standard deviation value of misclassification rate in the linearly separable data

DT	Pruned	RF50	RF500	Ada
2.35	0.97	0.33	0.32	0.31
(0.01)	(0.006)	(0.003)	(0.003)	(0.006)

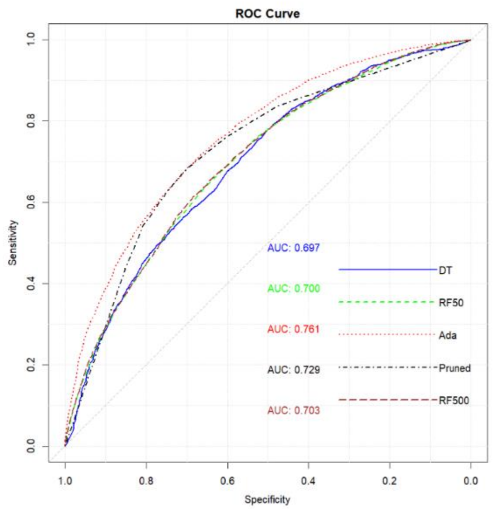
figure 14



In 1000 times simulation, DT seems has a little bit higher inaccuracy rate compare with other algorithms. We need to take care that although DT has strong interpretability, other algorithms may have more accurate prediction results in a very simple dataset. Besides that, the decision tree also has a larger standard deviation. In other words, DT can be more unstable than other algorithms as well. This issue will be augmented in the following cases 2 and case3. Comparing my result with the result from (Utami, I.T., et al, 2014), the misclassification rate of my decision tree is much larger than the paper's one, which is 0.44%. I think there are three main reasons. Firstly, the decision tree in the (Utami, I.T., et al, 2014) is probably the pruned tree instead of the full tree. In contrast, when I conduct the simulation, I forced the tree to grow as big as it can. In other words, there is no penalized term in my decision tree model. Furthermore, the misclassification rate of DT on paper is even lower than that of my pruned tree. My guessing is because I only take the complexity parameter into account, but there are still other parameters, such as min split and max depth, that can be applied to pruning. Therefore, the misclassification rate can decrease further. Secondly, probably because we do not have the same seed, so the outcome is different. Thirdly, the authors iterate it for 5000 times but I iterate 1000 times, which may also cause the difference. Additionally, my random forest prediction outcome is similar to the result in the paper. There is only a 0.01% difference between my rf50 and bagging 50 in the paper and roughly 0.1 % difference between rf500 and bagging 500 in the paper. Finally, Adaboost has the best performance among these algorithms, which has only roughly a 0.31% misclassification rate on average.

c. The simulation result for case 2

figure 15



As we can observe from figure 15, DT clearly has the lowest AUC compared with other algorithms. In contrast, pruned trees and Adaboost have relatively good performance. Compare with the previous case (case1), the pruned tree seems to improve more when the dataset is complex. If a dataset is simple, then pruned tree cannot make a great progression from an unpruned tree and the accuracy rate may also be lower than other algorithms.

Additionally, Adaboost still performs very well in case two compare with other algorithms. The main drawback of Adaboost is that it is very sensitive to outliers. Because if the outliers are misclassified, then the weight of outliers would get larger. Fortunately, our dataset follows multivariate normal distribution, and the sigma is under control. Moreover, we have 50000 individuals, so I think the effect of outliers is mitigated. In other words, even if the weight of outliers goes larger, it is still very small in the end. Therefore, Adaboost still can have an excellent performance here.

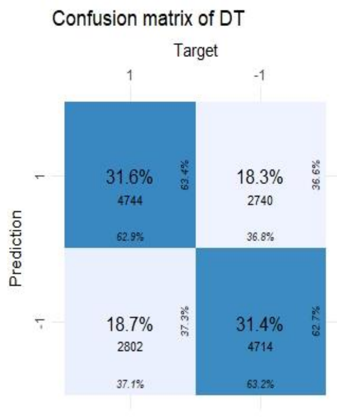


figure 16

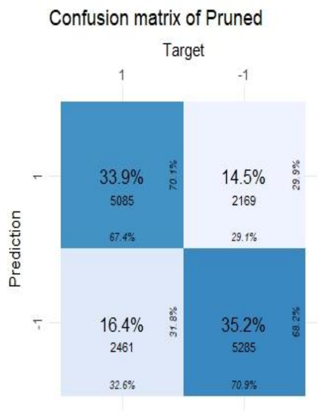


figure 17

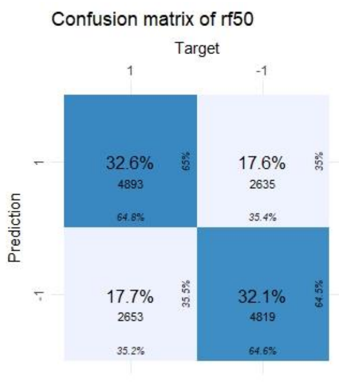


figure 18

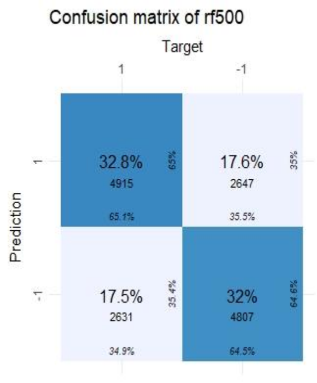


figure 19

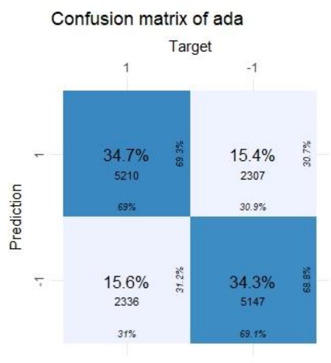


figure 20

To be more specific, pruned trees improve the full tree from roughly 63% accuracy rate to almost 70% accuracy rate. Conversely, the improvement of rf50 to rf500 is very limited, which only has a 0.1% accuracy rate increasing. Utami, I.T., et al (2014) also face a similar problem, which only increases by 0.12% as well. The main disadvantage of random forests is also very time demanding and when we make a prediction, effectiveness is also an important factor that we need to consider. Therefore, it seems not necessary need to used rf500 for insignificant improvement.

table 10

Percentage of average and standard deviation value of misclassification rate in the linearly non-separable data

DT	Pruned	RF50	RF500	Ada
79.06	33.18	36.41	36.20	35.39
(0.033)	(0.027)	(0.026)	(0.026)	(0.027)

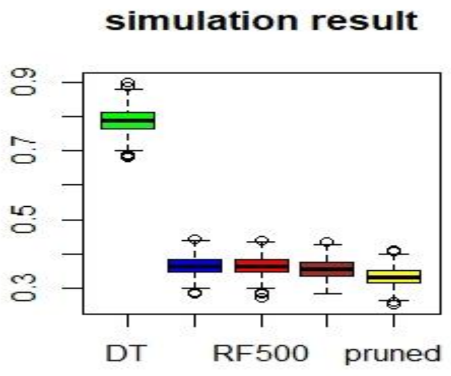


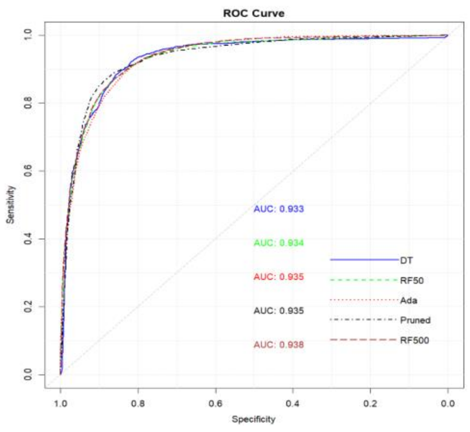
figure 21

From 1000 times simulation, the overfitting problem of DT makes it become a very inaccurate algorithm. My study also has a much higher inaccuracy rate than the one in (Utami, I.T., et al, 2014) in case2. Fortunately, pruning the tree let it become way much better, and the result is closed to the DT in the paper. Additionally, the misclassification rate of RF50 and RF500 is a little bit higher than the one in (Utami, I.T., et al ,2014). I think that except from control the number of trees in random forests, it is also possible to control the number of observations in the terminal node in each tree, which may help to further decrease the misclassification rate. Moreover, if I have more than two explanatory variables, it is very likely that the misclassification rate of random forest can decrease further by selecting partial predictors candidate.

Besides, comparing Adaboost with random forest, it is more accurate, but S.D. is also larger. I think if a few outliers in each simulation can be canceled, then Adaboost may have a better accuracy rate with lower S.D. than random forest.

d. The simulation result for case 3

figure 22



For case 3 once simulation, I set up 16666 observations for +1 class and 33332 for -1, so totally there are 49998 observations, then it can be compared with case1 and 2(with 50000 observations). From figure 22, we can notice that all algorithms have a similar result. The serious overfitting problem of DT seems did not show up in this simulation way. The problem of random forest that taken into account all predictors for each tree seems did not cause a serious problem here as well, otherwise, it should be worse than the Adaboost result significantly, because of a large number of observations, Adaboost also should not be too sensitive to outliers like it usually do. Additionally, in general, the performance of each algorithm is better than in the previous case. Therefore, the non-linear dataset classification problem seems can be solved by these algorithms.

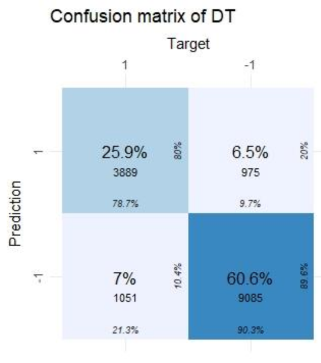


figure 23

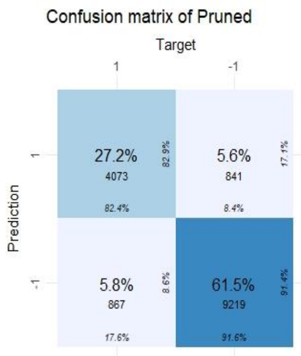


figure 24

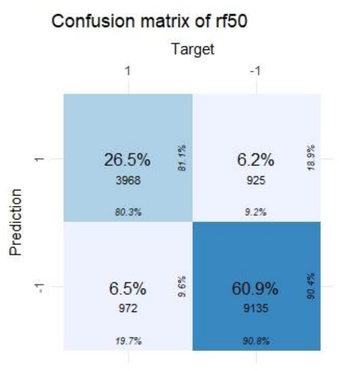


figure 25

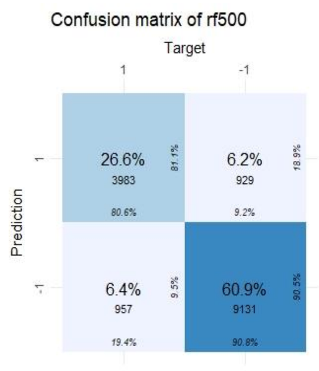


figure 26

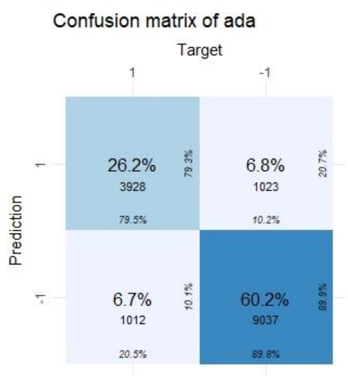


figure 27

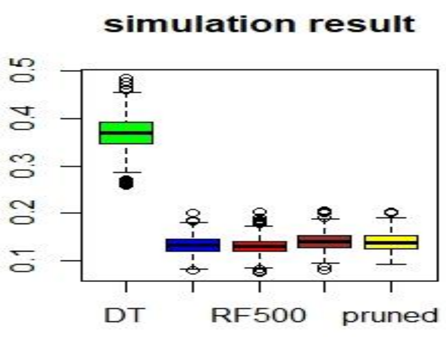
To be more specific, DT and pruned trees have 86.5% and 88.7% accuracy rates respectively. Moreover, rf50, rf500, and Ada have 87.4% ,87.5% and 86.4% accuracy rate respectively.

table 11

Percentage of average and standard deviation value of misclassification rate in the linearly non-separable data

DT	Pruned	RF50	RF500	Ada
36.91	13.92	13.20	13.07	14.09
(0.035)	(0.019)	(0.018)	(0.018)	(0.019)

figure 28



Next is the result from 1000 times simulation with 1200 observations each time. table 11 result is quite similar to the result from Utami, I.T., et al (2014). The difference between the pruned tree in my study and the DT result from Utami, I.T., et al (2014) is only 0.71% (so the difference in DT between my study and papers is still large) and the difference between bagging50, bagging500(from paper) and rf50, rf500(from my study) is only 0.43% and 0.39% respectively. Moreover, compared with case 2, seems all these algorithms perform well in linear non-separable data. Therefore, when facing real-world data which is non-linear separable, using a pruned tree, random forest and AdaBoost can be a clever choice. Among these algorithms, RF500 has the best performance, but its improvement from RF50 is still limited, so we can choose RF50 instead of RF500 to save time in case 3 similar datasets. Besides that, surprisingly, the pruned tree makes a progression from DT in case 3, which is different from the once simulation case. Therefore, we know that the overfitting problem is still an issue for case 3 1000 times simulation.

C. Conclusion

In conclusion, there are several insights that we can observe from the previous study. Firstly, comparing case1, case2 and case3, these algorithms can almost perfectly make a correct prediction in case one, which is a linearly separable dataset. The AUC of most algorithms is closed to 1 and the misclassification rate of most algorithms is below 1% as well. The second best performance case is in case 3 for all algorithms, which is a non-linear separable dataset. Finally, case2 has the worst performance, which is a linear non-perfectly separable dataset. Secondly, the accuracy rate of my decision tree is usually much lower than the one on paper in every case. I think it is because the tree in the paper is pruned tree, and the pruned tree makes a great progression from the unpruned tree in each case 1000 times simulation. Thirdly, in case1 and case2, my accuracy rate of the pruned tree is even worse than the DT in the paper. I think the main reason is that I only take the complexity parameter into account when pruning, but there are still many other parameters that can be used to prune the tree to increase the accuracy. Fourthly, in the case that dataset is hard to fit, the pruned tree seems to perform better compared with other algorithms. For instance, case 1 is a relatively easy fit dataset, but the pruned tree did not perform very well compare with boosting and bagging method. Contrary, case 2 is a relatively hard fit dataset, but the pruned tree has the best prediction outcome and makes a great progression from the unpruned tree. In other words, in a simple dataset with many observations, probably it is less important to prune the tree but for a dataset like a case 2 and only 840 training observations (70% of 1200), it becomes very necessary. Fifthly, Adaboost has a stable performance among these algorithms, because I think the drawbacks of AdaBoost, which is sensitive to outliers, did not be exposed in these three cases. Each case has 50000 observations and with the sigma that is under control, so the effects from the outliers and noise are mitigated. Sixthly, in all cases, the improvement of rf500 from rf50 is very limited. Therefore, it seems unnecessary to apply rf500 in this simulation study to save some calculation times. Seventhly, the overfitting problem seems more serious in the 1000 times simulation than once simulation with 50000 observations. Therefore, the improvement of the pruned tree is more significant in the 1000 times simulation. Eighthly, it is hard to say which algorithm is the best. In general, an unpruned tree has the worst performance. A pruned tree would have better performance in the complex dataset. Besides, because the quality of the dataset in each experiment is good, so the performance of Adaboost is also stable and accurate and the performance of random forest is also accurate and stable in each case.

D. Limitation

1. There are only two predictors, so the random forest cannot randomly choose a part of the candidate predictors for a split.
2. There is no comparison of different stumps for Adaboost.
3. If the outliers are eliminated, then the Adaboost should have better performance in each case.
4. The authors iterate the simulations 5000 times but I only iterate 1000 times.
5. There are many other pruning parameters, but only cp is taken into account here.
6. There are other pruning parameters for the random forest, but only the number of a tree is taken into account here.

E. Appendix

The way I created figure 1 is using dgp3 to run the random forest from 10, 20…..to 2000 iteratively and the code is like the following.

```r
data_test <- dgp_03(600)
sample <- sample.int(n = nrow(data_test), size = floor(.7*nrow(data_test)),
replace = F)
train <- data_test[sample, ]
test  <- data_test[-sample, ]
```

```r
rf_outcome <- c()
accuracy_rf <- c()
for (i in 1:200){
  rf_train <- randomForest(factor(y) ~ . , data = train,
                              mtry = 2 ,ntree = i*10)# take all the predicto
rs as sampled
  rf_predict<- predict(rf_train,test)
  for (k in 1:540){
    if (rf_predict[k] == test[k,"y"]){
      accuracy_rf[k] <- 1}
    else{accuracy_rf[k] <- 0}
  }
  rf_outcome[i] <- mean(accuracy_rf)
}
```

Besides, the following is the Simulation process and case2, and case1 have similar code.

```r
for (t in 1:T_iter){
  data <- dgp_03(400)
  sample <- sample.int(n = nrow(data), size = floor(.7*nrow(data)), replace
= F)
  train <- data[sample, ]
  test  <- data[-sample, ]
  #Decision Tree
  rpart_train <- rpart(y ~ .,data = train, control = c(cp = 0, minsplit = 0)
)
  rpart_predict <- predict(rpart_train,test)
  #Random Forest_50
  rf_train_50 <- randomForest(factor(y) ~ . , data = train,
                              mtry = 2 ,ntree = 50)# take all the predictors
 as sampled
  rf_predict_50 <- predict(rf_train_50,test)
  #Random Forest_500
  rf_train_500 <- randomForest(factor(y) ~ . , data = train,
                              mtry = 2 ,ntree = 500)# take all the predicto
rs as sampled
  rf_predict_500 <- predict(rf_train_500,test)
  #Adaboost
  train$y <- as.factor(train$y)#Need to change y as factor in advance
```

```r
ada_train <- boosting(y ~ ., train, boos=TRUE, mfinal=100)#Create 100 tree
 and using the weight to drawn
ada_predict <- predict(ada_train,test)
ada_predict$class
#Pruned Tree
d.tree.param <- makeClassifTask(
  data = train,
  target="y")

param_grid_multi <- makeParamSet(
  makeNumericParam("cp", lower = 0.001, upper = 0.01))

control_grid = makeTuneControlGrid()
resample = makeResampleDesc("CV", iters = 2L)
measure = acc

dt_tuneparam <- tuneParams(learner="classif.rpart",
                           task=d.tree.param,
                           resampling = resample,
                           measures = measure,
                           par.set=param_grid_multi,
                           control=control_grid,
                           show.info = TRUE)
best_parameters = setHyperPars(
  makeLearner("classif.rpart"),
  par.vals = dt_tuneparam$x
)


best_model = train(best_parameters, d.tree.param)
test$y <- as.factor(test$y)
d.tree.mlr.test <- makeClassifTask(
  data=test,
  target="y")
results <- predict(best_model, task = d.tree.mlr.test)$data
accuracy_prune <- mean(results$response == results$truth)

accuracy_tree <- c()
```

```r
accuracy_rf_50 <- c()
accuracy_rf_500<-c()
accuracy_ada<-c()


for (i in 1:360){
  if (rpart_predict[i] == test[i,'y']){
    accuracy_tree[i] <- 1}
  else{accuracy_tree[i] <- 0
  }
  if (rf_predict_50[i] == test[i,"y"]){
    accuracy_rf_50[i] <- 1}
  else{accuracy_rf_50[i] <- 0}
  if (ada_predict$class[i] == test[i,"y"]){
    accuracy_ada[i] <- 1}
  else{accuracy_ada[i] <- 0}
  if (rf_predict_500[i] == test[i,"y"]){
    accuracy_rf_500[i] <- 1}
  else{accuracy_rf_500[i]<-0}
}

accuracy_rate_tree <- mean(accuracy_tree)
mis_class_tree_03[t] <- 1-accuracy_rate_tree
accuracy_rate_rf <- mean(accuracy_rf_50)
mis_class_rf_50_03[t] <- 1-accuracy_rate_rf
accuracy_rate_ada <- mean(accuracy_ada)
mis_class_ada_03[t] <- 1-accuracy_rate_ada
accuracy_rate_rf_500 <- mean(accuracy_rf_500)
mis_class_rf_500_03[t] <- 1-accuracy_rate_rf_500
mis_class_prune_03[t] <- 1-accuracy_prune
}
```

F. Reference

Utami, I. T. , Sartono, B. , & Sadik, K. (2014). Comparison of Single and Ensemble Classifiers of Support Vector Machine and Classification Tree. Journal of Mathematical Sciences and Applications, 2(2), 17-20.

Breiman,L. (2001). Random Forest. Machine Learning, 45, 5-32.

https://doi.org/10.1023/A:1010933404324.

Freund,Y., Schapire,R.E. (1996). Experiment with a New Boosting Algorithm. Machine Learning: Proceedings of the Thirteenth International Conference.

Khoshgoftaar, T.M., Hulse, J.V., & Napolitano, A. (2011). Comparing Boosting and Bagging Techniques With Noisy and Imbalanced Data. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 41,* 552-568.

Therneau, T.M., Atkinson, E.J., & Foundation, M. (2022). An Introduction to Recursive Partitioning Using RPART Routines. https://cran.rproject.org/web/packages/rpart/vignettes/longintro.pdf

Sam, T. (2019). How does Decision Tree Make Decisions-The sample logic and math behind machine learning algorithm. https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8

Kurama, V. (2020). A Guide to Adaboost: Boosting To Save The Day. https://blog.paperspace.com/adaboost-optimizer/.

Iba, W., & Langley, P. (1992). Induction of one-level decision trees. Proceedings of the Ninth International Conference on Machine Learning (pp. 233–240). Morgan Kaufmann Publishers.