Kernel Regression Tree

Junting Ren

Columbia University, Department of Biostatistics, New York, USA
jr3755@cumc.columbia.edu

1 Introduction

Regression trees provide quite simple and interpretable regression models with reasonable accuracy. However, these methods are known for their instability. Weiss & Indurkhya (1995) used a nearest neighbor approximation in their regression rules inductive system. Quinlan (1992) and Karalic (1992) have used linear models in regression tree leaves. Torgo (1999) used kernel regression models on data in the corresponding leaves to predict outcome (Local regression tree). Torgos method showed superiority compared to the traditional method that use mean in the leaves to predict outcome, but it also requires significantly more local memory and computational time. However, through some searching, I did not find articles that compare it with random forest and regression tree bagging. Furthermore, existing R packages do not provide kernel regression predictions for regression trees.

The intuition for this method is that the traditional tree algorithm only used the mean of the outcome of the corresponding leaf's data to do prediction. The information of the variables' are not utilized in the final prediction. Moreover, some variables are not used in the spliting, but they can still provide some information for prediction. Therefore, we would like to use all the predictors available in the final prediction in the leaf to get a more robust prediction. By preserving the predictors' data in the leaf, given a new data point, we can calculate the distances(based on different predictors) to the training data points, and using this distance to weight the outcome in the final prediction.

2 Method

2.1 Algorithm

The learning stage The learning stage of our method consists of developing a regression tree. The algorithm we use is similar to the one described by Breiman et al. (1984) employed in CART. We grow a binary tree with the goal of minimizing the mean squared error (MSE) obtained by the average Y value using a formulation similar to the one given in Equation 1. This corresponds to dividing the input space into regions with similar Y values. Splits on continuous variables are done by choosing the cut-point value that maximizes the gain of MSE. Splits

on discrete variables consist of finding the subset of values that maximizes the referred gain.

$$Err(s,t) = Err(t) - P_L \times Err(t_L) - P_R \times Err(t_R) \qquad (1)$$

The package for building we used is the 'tree' package. More details can be found here: https://cran.r-project.org/web/packages/tree/tree.pdf

The prediction stage Given a query point q, we run it through the tree until a leaf node is reached. At this stage instead of using the average Y value of the training samples on the leaf, we used k-nearest neighbor weighted average to do the prediction. The details of k-nearest neighbor weighted average algorithm can be found from Hechenbichler (2004).

Kernel Regression Tree Prediction:
Input: Regression tree *tree*, new data points x_i, training data *train*, number of nearest neighbor k, type of kernel $K(\cdot)$
Output: A list of prediction value $\hat{y}_i$
Steps:
1. Find the terminal leaf index for each new data point j. Function: f_node
2. Split the training data set into different terminal leaves L_j. Function: f_train_leaves
3. Calculate K-nearest neighbor weighted average by using x_i and its corresponding leafs training data L_j. Function: f_kernel

---

The detailed code of this algorithm please refer to our R code in the appendix.

2.2 Data simulation

From the same 10 predictors, four different addictive models were used to generate the responses.

$$i = 1, 2, 3, ..., 10 \qquad x_i \sim N(0,1) \qquad e \sim N(0,1)$$

1. Outcome generated with all linear predictors

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * x_3 + b_4 * x_4 + b_5 * x_5 + b_6 * x_6 + b_7 * x_7 + b_8 * x_8 + b_9 * x_9 + b_{10} * x_{10} + e$$

2. Outcome generated with portion of linear predictors

$$y = b_0 + b_1 * x_1 + b_3 * x_3 + b_4 * x_4 + b_6 * x_6 + b_7 * x_7 + b_8 * x_8 + b_{10} * x_{10} + e$$

3. Outcome generated with quadratic predictors

$$y = b_0 + b_1 * x_1^2 + b_2 * x_2^2 + b_3 * x_3^2 + b_4 * x_4^2 + b_5 * x_5^2 + b_6 * x_6^2 + b_7 * x_7^2 + b_8 * x_8^2 + b_9 * x_9^2 + b_{10} * x_{10}^2 + e$$

4. Outcome generated with portion of quadratic predictors

$$y = b_0 + b_1 * x_1^2 + b_2 * x_2^2 + b_4 * x_4^2 + b_6 * x_6^2 + b_7 * x_7^2 + b_8 * x_8^2 + b_{10} * x_{10}^2 + e$$

All four models were simulated 10 times and each with 400 sample points(half for training and half for testing). The simulated data was put into boosting, global k-nearest neighbor average, random forest, bagging, tree and kernel regression tree to calculate the testing mean square error. For models that are generated with portions of the predictors, all ten predictors are input into the predictions.

2.3 Real data sets

All our real data sets were obtained from UCI Machine Learning Repository.Housing Values in Suburbs of Boston, Breast Cancer Wisconsin (Prognostic) Data Set, Airfoil Self-Noise Data Set and Concrete Compressive Strength Data Set were used for testing the accuracy of our models. Only continous outcomes were used in this testing. For more details of the data sets, please refer to http://archive.ics.uci.edu/ml/index.php

3 Results

As we can see in figure 1, Kernel Regression Tree only performed slightly better than the traditional parametric tree method for each simulated data sets. Overall, boosting had the greatest accuracy. Interestingly, all methods performed better when the model is generated with only selective predictors.

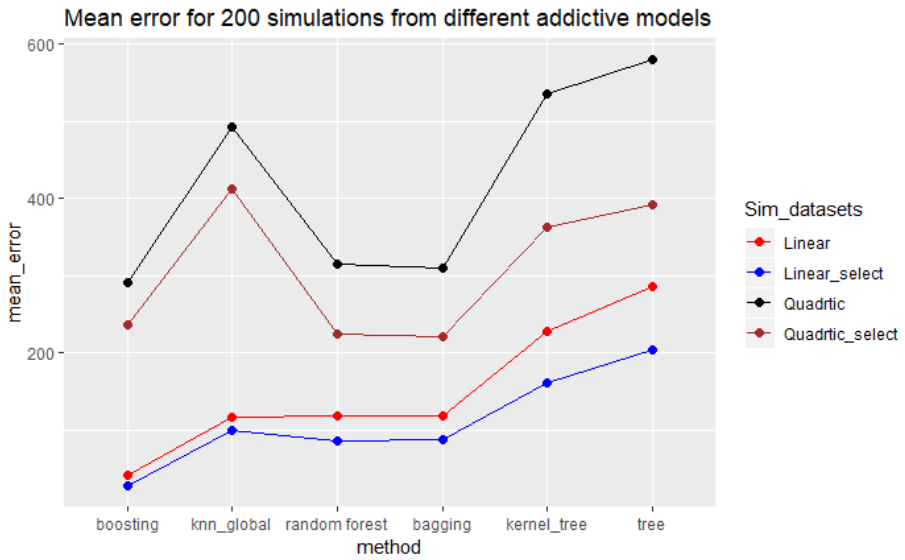


Fig. 1. Mean MSE(mean square error) of simulated data sets for different methods

From figure 2, kernel regression tree performed slightly better than traditional

tree and global k-nearest neighbor weighted average on three data sets. But for the breast cancer recurrence free time prediction, both kernel regression tree and traditional tree performed significantly worse than the other three methods.

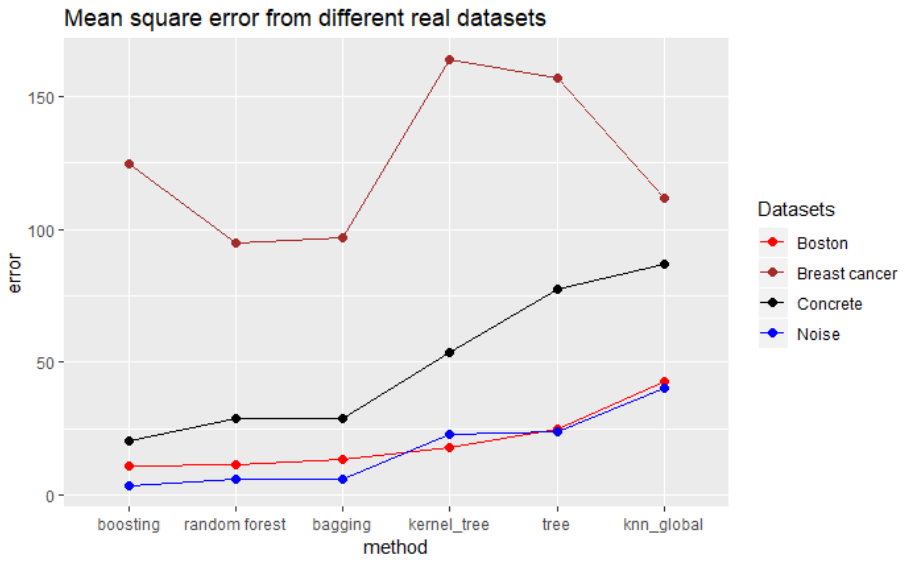


Fig. 2. Mean square error of real data sets for different methods

4 Discussion

From the results, it is obvious that Kernel regression tree is only a bit improvement from the traditional tree. When consider the significant amount of prediction time increase compared to the relatively small accuracy gain, it is very hard to conclude that this new method is superior than traditional tree. Moreover, other ensemble tree methods perform significantly better than kernel regression tree with the same or even less computing time. The only advantage kernel regression tree provides is that it preserves interpretability and relative good accuracy.

It is still possible to implement kernel regression tree in specific situations. For example, in medical or scenarios that requires people to understand, many decisions are rule based. Tree algorithm provides a good interpretability and we could preserve this it and increase the accuracy by using kernel regression tree. However, this method could only be used under relatively simple situations where variables don't interact with each other much. If the situation is complex, like in predicting breast cancer recurrence free time, kernel regression tree method still lacks the model complexity to do a good prediction.

An interesting observation is that all the methods perform slightly better with the model generated by only portion of the data set. But the prediction is made with all the predictors. I speculate that this is because tree method assumes that some predictors are useless and only use the most efficient predictors instead.

5 Conclusion

Kernel regression tree performs slightly better than traditional tree method but performs worse than other ensemble tree methods. More work is needed to find the specific scenario where kernel regression tree is practical when considering the computation time it consumes

Kernel Regression Tree

Junting Ren Uni:jr3755

November 29, 2018

Writing a prediction function

```
tree_predict_kernel = function(tree_in, new_data, train, k, kernel){#train: with response in the last c
  #Cleaning the frame for the tree
  frame = tree_in$frame
  frame$var = as.character(frame$var)
  frame = cbind(frame,data.frame(split_l = frame$splits[,1]))
  frame = frame %>% select(-splits) %>% rownames_to_column('node_n') %>%
    mutate(split_l = str_remove(split_l,'<'),
           split_l = as.numeric(split_l), node_n = as.numeric(node_n)) #For numeric variables only
  response_index = ncol(train)

  #Function for finding the final leaf(number) number (recursive) for a new sample point x_s, starts wi
  f_node = function(x_s){
    x_s = x_s
    f_node_inner = function(node_num){
      node = frame %>% filter(node_n == node_num)
      if(is.na(node$split_l)){
        return(node_num)
      }else{
        if(x_s[,node$var] < node$split_l){
          node_num = node_num * 2
          f_node_inner(node_num)
        }else{
          node_num = node_num * 2 + 1
          f_node_inner(node_num)
        }
      }
    }
    f_node_inner(1)
  }


  #Function for spliting the training data into leaf space (return list of subsets of data for all leav
  f_train_leaves = function(train){
    leaf_n_list = frame %>% filter(is.na(split_l)) %>% pull(node_n)
    #function for dividing the data
    #input: one leaf number; output: the subset of the training data for the leaf
    f_divide = function(leaf_n){
      split_val = c()
      split_var = c()
      leaf_save = c(leaf_n)
      train_subset = train
      f_divide_inner = function(leaf_n){
        if(is.even(leaf_n)){
          leaf_n = leaf_n/2
```

```
        if(leaf_n!=1){
          leaf_save <<- c(leaf_n, leaf_save)
        }
        split_node = frame %>% filter(node_n == leaf_n)
        split_val <<- c(split_node %>% pull(split_l), split_val)
        split_var <<- c(split_node %>% pull(var), split_var)
        f_divide_inner(leaf_n)
      }else{
        if(leaf_n == 1){
          for(i in 1:length(split_var)){
            if(is.even(leaf_save[i])){
              train_subset <<- train_subset %>% filter((!!as.name(split_var[i])) < split_val[i])
            }else{
              train_subset <<- train_subset %>% filter((!!as.name(split_var[i])) >= split_val[i])
            }
          }
          train_subset
        }else{
          leaf_n = (leaf_n-1)/2
          if(leaf_n!=1){
            leaf_save <<- c(leaf_n, leaf_save)
          }
          split_node = frame %>% filter(node_n == leaf_n)
          split_val <<- c(split_node %>% pull(split_l), split_val)
          split_var <<- c(split_node %>% pull(var), split_var)
          f_divide_inner(leaf_n)
        }
      }
    }
    f_divide_inner(leaf_n)
  }
  lapply(leaf_n_list, f_divide)
}

#Function for running kernel for every sample point using the leaf-subset data (return predicted valu
f_kernel = function(x_s){
  node_num = f_node(x_s)
  leaf = leaves[[as.character(node_num)]]
  if(nrow(leaf) > k){
    kknn <- kknn(paste(names(leaf)[response_index],'~','.'), leaf, x_s, k = k, kernel = kernel, scale
    fitted(kknn)
  }else{
    kknn <- kknn(paste(names(leaf)[response_index],'~','.'), leaf, x_s, k = nrow(leaf)-1, kernel = ke
    fitted(kknn)
  }
}



#Starting regression process
leaves = f_train_leaves(train)#get the training leaves
names(leaves) = frame %>% filter(is.na(split_l)) %>% pull(node_n)#Put leaf number for each training l
as.numeric(by(new_data,1:nrow(new_data), f_kernel))
```

```
}
```

Simulation study

function for getting the error and sd for one simulate dataset from different methods

```
#The last col of the sim_data data is the outcome we want to predict.
apply_methods_result = function(sim_data){
  #Spilit dataset into training and testing: 50:50
  set.seed(1)
  train_index = sample (1: nrow(sim_data), nrow(sim_data )/2)
  train = sim_data[train_index,]
  #getting the formular for other regressions
  formu = as.formula(paste(names(train)[ncol(train)],'~','.'))
  #building a tree
  tree_in=tree(formu,train)
  summary (tree_in )
  y = sim_data[-train_index,ncol(sim_data)]
  new_data = sim_data[-train_index,-ncol(sim_data)]#for testing
  #Kernel tree
  y_kernel = tree_predict_kernel(tree_in, new_data, train, k = 6, kernel =  "cos" )
  error_kernel = mean((y - y_kernel)^2)
  SD_kernel = sd((y - y_kernel)^2)
  #Normal tree
  y_tree = predict(tree_in, new_data)
  error_tree = mean((y - y_tree)^2)
  SD_tree = sd((y - y_tree)^2)


  #Bagging
  set.seed(1)
  bag.boston =randomForest(formu,data = train ,mtry=ncol(new_data), importance = TRUE)
  y_bagging = predict (bag.boston ,newdata = new_data)
  error_bag = mean((y - y_bagging)^2)
  SD_bag = sd((y - y_bagging)^2)
  #Random forest
  set.seed(1)
  rf.boston =randomForest(formu,data=train, mtry=6, importance =TRUE)
  y_rf = predict (rf.boston ,newdata = new_data)
  error_rf = mean((y - y_rf)^2)
  SD_rf = sd((y - y_rf)^2)
  #Boosting
  set.seed(1)
  boost.boston =gbm(formu,data=train, distribution= "gaussian",n.trees =5000 , interaction.depth =4)
  y_boosting =predict (boost.boston ,newdata = new_data, n.trees=5000)
  error_boosting = mean((y - y_boosting)^2)
  SD_boosting = sd((y - y_boosting)^2)
  #kernel knn global
  kknn <- kknn(formu, train, new_data, k = 6, kernel = 'cos', scale = F)
  y_knn_global = fitted(kknn)
```

```
  error_knn_global = mean((y - y_knn_global)^2)
  SD_knn_global = sd((y - y_knn_global)^2)
  #Result
  result = tibble(method = c('kernel_tree', 'tree', 'bagging', 'random forest', 'boosting', 'knn_global
                         error = c(error_kernel, error_tree, error_bag, error_rf, error_boosting, error_k
                         SD = c(SD_kernel,SD_tree,SD_bag,SD_rf,SD_boosting,SD_knn_global))
  return(result)
}
```

Linear model generated dataset

Outcome generated with all linear predictors

```
#Simulation for the linear model
simfun <- function(b_0=3,b_1=10,b_2=8,b_3=6,b_4=4,b_5=2,b_6=-10,b_7=-8,b_8=-6,b_9=-4,b_10=-2,
                     n=200,x1.sd=1,x2.sd=1,e.sd=1) {
   x1 <- rnorm(n, mean=0, sd=x1.sd)
   x2 <- rnorm(n, mean=0, sd=x2.sd)
   x3 <- rnorm(n, mean=0, sd=x1.sd)
   x4 <- rnorm(n, mean=0, sd=x2.sd)
   x5 <- rnorm(n, mean=0, sd=x1.sd)
   x6 <- rnorm(n, mean=0, sd=x2.sd)
   x7 <- rnorm(n, mean=0, sd=x1.sd)
   x8 <- rnorm(n, mean=0, sd=x2.sd)
   x9 <- rnorm(n, mean=0, sd=x1.sd)
   x10 <- rnorm(n, mean=0, sd=x2.sd)
   e <-  rnorm(n, mean=0, sd=e.sd)
   y1 <- b_0+b_1*x1+b_2*x2+b_3*x3+b_4*x4+b_5*x5+b_6*x6+b_7*x7+b_8*x8+b_9*x9+b_10*x10+e
   data.frame(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,y1)
}

#Simulate data
n = 10
sim_datas = vector("list", n)
for(i in 1:n){
  sim_data = simfun(n = 400)
  sim_datas[[i]] = sim_data
  i = i + 1
}

#applying all the results to the simluated datas
results = lapply(sim_datas, apply_methods_result)
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
#bind them into one dataframe
result_linear_sim = do.call(rbind,results) %>% group_by(method) %>%
  summarise(mean_error = mean(error), mean_SD = mean(SD)) %>% mutate(method = forcats::fct_reorder(meth
```

Outcome generated with all non-linear predictors

```
#Simulation for the non-linear model
simfun <- function(b_0=3,b_1=10,b_2=8,b_3=6,b_4=4,b_5=2,b_6=-10,b_7=-8,b_8=-6,b_9=-4,b_10=-2,
                     n=200,x1.sd=1,x2.sd=1,e.sd=1) {
   x1 <- rnorm(n, mean=0, sd=x1.sd)
   x2 <- rnorm(n, mean=0, sd=x2.sd)
   x3 <- rnorm(n, mean=0, sd=x1.sd)
   x4 <- rnorm(n, mean=0, sd=x2.sd)
   x5 <- rnorm(n, mean=0, sd=x1.sd)
   x6 <- rnorm(n, mean=0, sd=x2.sd)
   x7 <- rnorm(n, mean=0, sd=x1.sd)
   x8 <- rnorm(n, mean=0, sd=x2.sd)
   x9 <- rnorm(n, mean=0, sd=x1.sd)
   x10 <- rnorm(n, mean=0, sd=x2.sd)
   e <-  rnorm(n, mean=0, sd=e.sd)
   y1 <- b_0+b_1*(x1^2)+b_2*(x2^2)+b_3*(x3^2)+b_4*(x4^2)+b_5*(x5^2)+b_6*(x6^2)+b_7*(x7^2)+b_8*(x8^2)+b_
   data.frame(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,y1)
}

#Simulate data
n = 10
sim_datas = vector("list", n)
for(i in 1:n){
  sim_data = simfun(n = 400)
  sim_datas[[i]] = sim_data
  i = i + 1
}

#applying all the results to the simluated datas
results = lapply(sim_datas, apply_methods_result)

#bind them into one dataframe
result_nonlinear_sim = do.call(rbind,results) %>% group_by(method) %>%
  summarise(mean_error = mean(error), mean_SD = mean(SD)) %>% mutate(method = forcats::fct_reorder(metho
```

Outcome generated with selective linear predictors

```
#Simulation for the selective linear model
simfun <- function(b_0=3,b_1=10,b_2=8,b_3=6,b_4=4,b_5=2,b_6=-10,b_7=-8,b_8=-6,b_9=-4,b_10=-2,
                     n=200,x1.sd=1,x2.sd=1,e.sd=1) {
   x1 <- rnorm(n, mean=0, sd=x1.sd)
   x2 <- rnorm(n, mean=0, sd=x2.sd)
   x3 <- rnorm(n, mean=0, sd=x1.sd)
   x4 <- rnorm(n, mean=0, sd=x2.sd)
   x5 <- rnorm(n, mean=0, sd=x1.sd)
   x6 <- rnorm(n, mean=0, sd=x2.sd)
   x7 <- rnorm(n, mean=0, sd=x1.sd)
   x8 <- rnorm(n, mean=0, sd=x2.sd)
   x9 <- rnorm(n, mean=0, sd=x1.sd)
   x10 <- rnorm(n, mean=0, sd=x2.sd)
   e <-  rnorm(n, mean=0, sd=e.sd)
```

```
   y1 <- b_0+b_1*x1+b_3*x3+b_4*x4+b_6*x6+b_7*x7+b_8*x8+b_10*x10+e
   data.frame(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,y1)
}

#Simulate data
n = 10
sim_datas = vector("list", n)
for(i in 1:n){
  sim_data = simfun(n = 400)
  sim_datas[[i]] = sim_data
  i = i + 1
}

#applying all the results to the simluated datas
results = lapply(sim_datas, apply_methods_result)

#bind them into one dataframe
result_linear_select_sim = do.call(rbind,results) %>% group_by(method) %>%
  summarise(mean_error = mean(error), mean_SD = mean(SD)) %>%  mutate(method = forcats::fct_reorder(meth
```

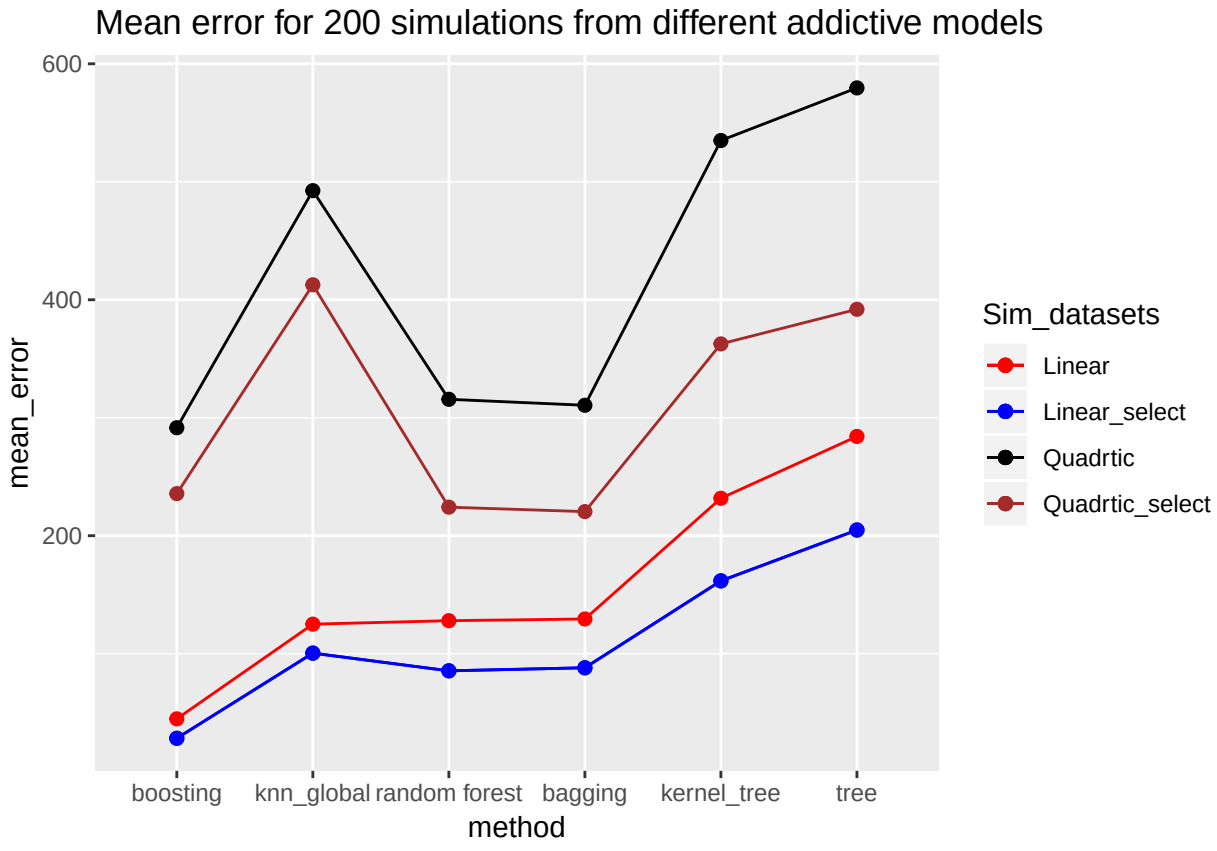
Outcome generated with selective non-linear predictors

```
#Simulation for the selective linear model
simfun <- function(b_0=3,b_1=10,b_2=8,b_3=6,b_4=4,b_5=2,b_6=-10,b_7=-8,b_8=-6,b_9=-4,b_10=-2,
                     n=200,x1.sd=1,x2.sd=1,e.sd=1) {
   x1 <- rnorm(n, mean=0, sd=x1.sd)
   x2 <- rnorm(n, mean=0, sd=x2.sd)
   x3 <- rnorm(n, mean=0, sd=x1.sd)
   x4 <- rnorm(n, mean=0, sd=x2.sd)
   x5 <- rnorm(n, mean=0, sd=x1.sd)
   x6 <- rnorm(n, mean=0, sd=x2.sd)
   x7 <- rnorm(n, mean=0, sd=x1.sd)
   x8 <- rnorm(n, mean=0, sd=x2.sd)
   x9 <- rnorm(n, mean=0, sd=x1.sd)
   x10 <- rnorm(n, mean=0, sd=x2.sd)
   e <-  rnorm(n, mean=0, sd=e.sd)
   y1 <- b_0+b_1*(x1^2)+b_3*(x3^2)+b_4*(x4^2)+b_6*(x6^2)+b_7*(x7^2)+b_8*(x8^2)+b_10*(x10^2)+e
   data.frame(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,y1)
}

#Simulate data
n = 10
sim_datas = vector("list", n)
for(i in 1:n){
  sim_data = simfun(n = 400)
  sim_datas[[i]] = sim_data
  i = i + 1
}

#applying all the results to the simluated datas
results = lapply(sim_datas, apply_methods_result)
```

```r
#bind them into one dataframe
result_nonlinear_select_sim = do.call(rbind,results) %>% group_by(method) %>%
  summarise(mean_error = mean(error), mean_SD = mean(SD)) %>% mutate(method = forcats::fct_reorder(metho
```

Plotting for simulation data

```r
ggplot() +
  geom_point(data = result_linear_sim, aes(x = method, y = mean_error, color = 'Linear'), size = 2) +
  geom_line(data = result_linear_sim, aes(x = method, y = mean_error, group = 1, color = 'Linear')) +
  geom_point(data = result_linear_select_sim, aes(x = method, y = mean_error, color = 'Linear_select'),
  geom_line(data = result_linear_select_sim, aes(x = method, y = mean_error, group = 2,color = 'Linear_s
  geom_point(data = result_linear_select_sim, aes(x = method, y = mean_error, color = 'Linear_select'),
  geom_line(data = result_linear_select_sim, aes(x = method, y = mean_error, group = 2,color = 'Linear_s
  geom_point(data = result_nonlinear_sim, aes(x = method, y = mean_error, color = 'Quadrtic'), size = 2)
  geom_line(data = result_nonlinear_sim, aes(x = method, y = mean_error, group = 2,color = 'Quadrtic'))
  geom_point(data = result_nonlinear_select_sim, aes(x = method, y = mean_error, color = 'Quadrtic_selec
  geom_line(data = result_nonlinear_select_sim, aes(x = method, y = mean_error, group = 2,color = 'Quadr
  scale_colour_manual("Sim_datasets",
                      values = c("Linear"="red", "Linear_select"="blue",
                                 "Quadrtic"="black", "Quadrtic_select" = "brown")) +
  labs(title = "Mean error for 200 simulations from different addictive models")
```



Real datasets

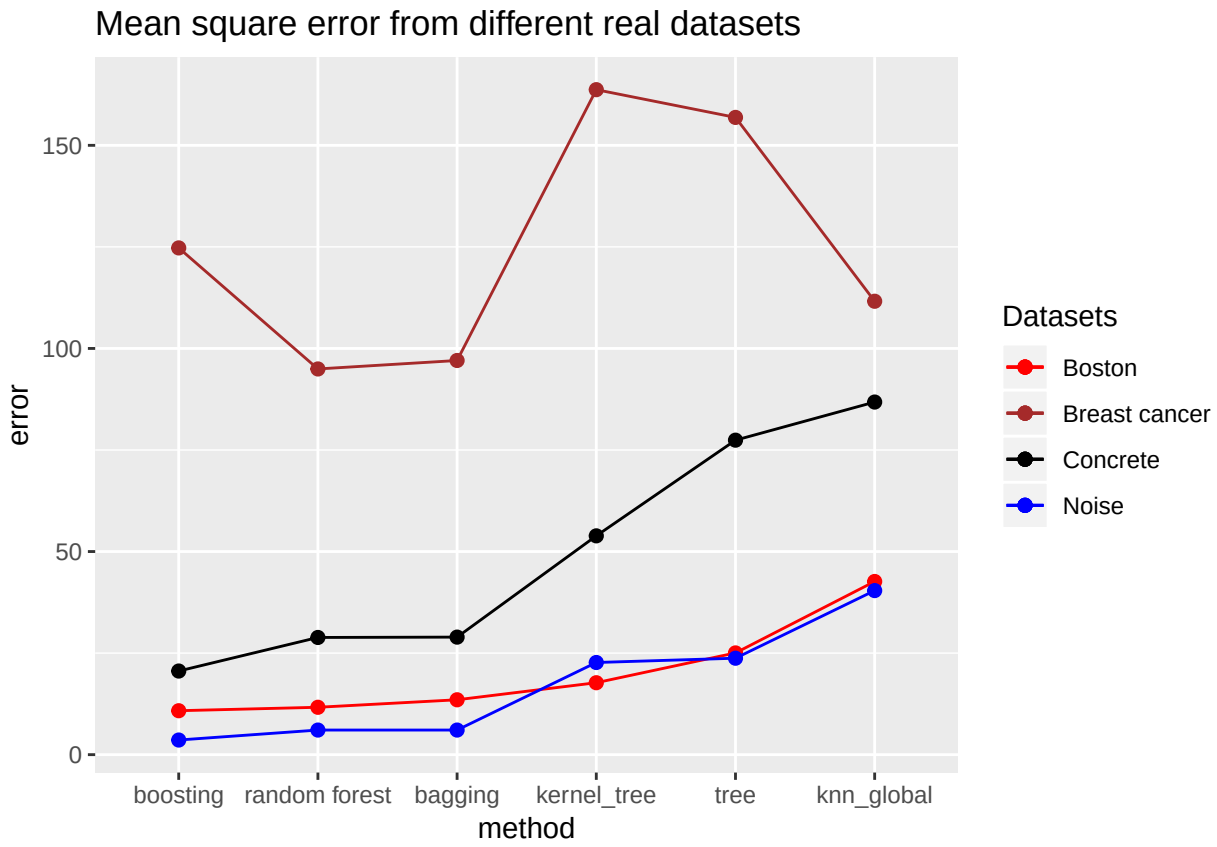
```
#Boston datase
result_boston = apply_methods_result(Boston) %>% mutate(method = forcats::fct_reorder(method, error))
#Airfoil Self-Noise Data Set
noise = read.table('http://archive.ics.uci.edu/ml/machine-learning-databases/00291/airfoil_self_noise.da
          header = F) # 5 predictors
result_noise = apply_methods_result(noise) %>% mutate(method = forcats::fct_reorder(method, error))
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within
## valid range
```

```
#Concrete Compressive Strength Data
concrete = read_excel("./data/Concrete_Data.xls", sheet = 1) %>% janitor::clean_names()
result_concrete = apply_methods_result(concrete) %>% mutate(method = forcats::fct_reorder(method, error
#Breast cancer recurrence
breast = read.table('http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/w
                    sep = ',') %>%
  select(-1,-2,-35, -V3, V3)
result_breast = apply_methods_result(breast) %>% mutate(method = forcats::fct_reorder(method, error)) %
```

ploting for the real datasets

```
ggplot() +
  geom_point(data = result_boston, aes(x = method, y = error, color = 'Boston'), size = 2) +
  geom_line(data = result_boston, aes(x = method, y = error, group = 1, color = 'Boston')) +
  geom_point(data = result_noise, aes(x = method, y = error, color = 'Noise'), size = 2) +
  geom_line(data = result_noise, aes(x = method, y = error, group = 2,color = 'Noise')) +
  geom_point(data = result_concrete, aes(x = method, y = error, color = 'Concrete'), size = 2) +
  geom_line(data = result_concrete, aes(x = method, y = error, group = 2,color = 'Concrete')) +
  geom_point(data = result_breast, aes(x = method, y = error, color = 'Breast cancer'), size = 2) +
  geom_line(data = result_breast, aes(x = method, y = error, group = 2,color = 'Breast cancer')) +
  scale_colour_manual("Datasets",
                      values = c("Boston"="red", "Noise"="blue",
                                 "Concrete"="black", "Breast cancer" = "brown")) +
  labs(title = "Mean square error from different real datasets")
```


Mean square error from different real datasets
150
100
50
0
error
boosting
random forest
bagging
kernel_tree
tree
knn_global
method
Datasets
Boston
Breast cancer
Concrete
Noise