ISEN 613 - Spring 2021
Course Project - Final Report

- Ridham Patoliya (730xxxxxxx)
- Karmit Shah (831xxxxxxx)
- Purva More (831xxxxxxx)
- Smriti Singh (630xxxxxxx)
- Prafull Deosarkar (630xxxxxxx)

Date : 4/29/2021

Guided by: Dr. Shahin Shahrampour

Executive Summary

Through the series of experiments, our goal was to present a model with the best prediction. There were many different algorithms to choose from starting from linear regression, subset selection, regularization to decision trees, and Random forests. But as the "No Free Lunch Policy" goes, each algorithm has its strengths and weaknesses, and we can use these properties to choose an algorithm and tune it to meet our goal.

The Objective:
Our goal is to obtain the best ***predictions*** from unseen data using different statistical learning algorithms.

The Problem:
The training data set for this project consists of 550 data points as rows and 8 columns as features. The test data consists of 218 data points and 8 columns as features. To understand the concepts of data analytics and explore various methodologies behind it, the team was tasked to build several models to predict the response for the data and select the best 3 amongst them.

Key Highlights:
We have researched extensively to learn about different statistical learning methods. Initially, we tried fitting simple linear models and simple decision trees. Because of non-linearity in data, we tried using models that can handle non-linearity better. We used different variations of decision trees and concluded that boosting is the best method that is aligned to our objectives. Tuning of hyper-parameters and Cross-Validation played an important role in the process. In our project, data analysis is done with R Statistical Software.

The Solution Procedure:

- Visualizing data and looking for correlations and non-linearities
- Recognition of suitable models
- Optimization of models
- Evaluation of optimized models using the same metric (test error on validation set)
- Choosing the best model

Results:
The 3 best models obtained with the lowest training error rates were MLR, Bagging and Boosting. On validation data, MLR, Bagging and Boosting resulted in a test error ***of 2.087, 0.346*** and ***0.125*** respectively. Boosting was chosen as the best model owing to its lowest error rate. The test data resulted in an MSE of **0.266** using the boosting model. (Caret package gives different results with different platforms. We are yet to figure out a way to manage that issue)

- Best model = ***Tree Boosting***
- Best test error for unseen data = ***0.266 (Model with Improvement)***
- Best test error on validation set = ***0.125***

This project was done in complete collaboration of all team members. Initially, all members came up with their best models and then best model was finalized after brainstorming.
Contribution from each team member (in alphabetical order):

- Karmit: •Technical analysis on MLR • Final editor of Phase I report • Improvement in boosting
- Prafull: •Technical analysis on Bagging • Final test results report • Executive summary
- Purva: •Technical analysis on Bagging • Final editor of Phase I & II report • Executive summary
- Ridham: •Technical analysis on Boosting • Model improvement report • Best model report
- Smriti: •Technical analysis on Boosting • Executive summary first draft • Best model report

Tree Boosting

Overview:

Here, we have summarized our implementation of tree boosting using **gbm** package. For tuning, we have used **caret** package.

Tree Boosting, if not controlled, can perform significantly well on training data and can even learn individual data points resulting in zero training mean square error. Therefore, it is imperative to tune the hyperparameters such that it does not overfit and can provide reasonable predictions for our test data. We have used 5-fold cross validation in to find optimum hyperparameters using grid search.

Data (D.train, D.test):

We performed cross-validation and tuning using training data and calculated MSE for the hold-out (30%) test data.

Feature Engineering and manipulating predictor space:

Decision trees, in general, captures interactions between predictors. Tree boosting further improves this performance by capturing interaction slowly as it goes from one child tree to another. Predictors that are rendered insignificant for linear regression have high relative influence in boosting. Therefore, when we tried removing these predictors, we obtained mediocre results.

Evaluation Metrics:

To compare tree boosting with other methods, we used test MSE. Our tuning algorithm uses cross validation errors to compare different boosted trees.

Hyperparameters:

There are 4 most important hyperparameters:

n.trees = Number of trees (Number of boosting iterations.) Very high n.trees results in overfitting.

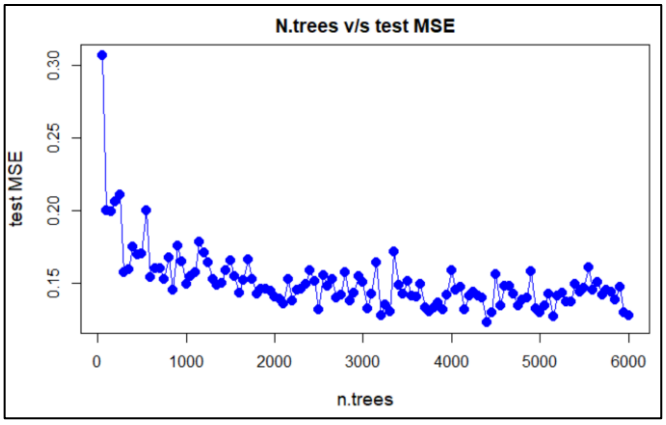
Interaction.depth (d) = Number of splits for all small trees. If interaction depth is more than 1, the algorithm captures interactions. High value of (**d**) can result in overfitting.

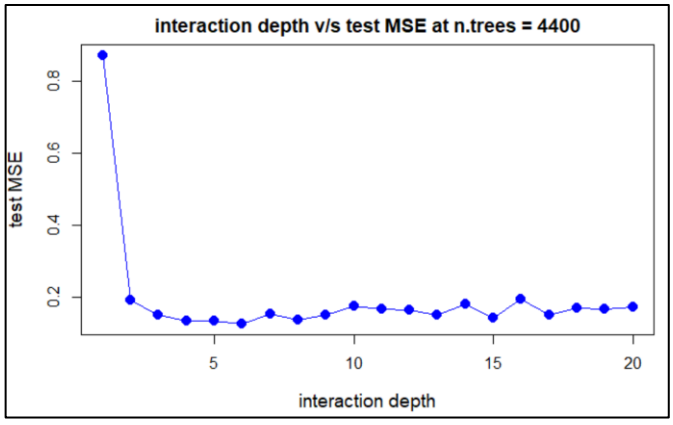
Shrinkage (λ) = Rate at which boosting algorithm learns sequentially through the small trees. Algorithm takes long to converge if **λ** is too small because we need high n.trees for good results. If **λ** is large, algorithm can converge quickly for small n.trees but the results are not reliable because of possibility of overshooting the optima.

n.minobsinnod = Number of minimum observations in a node. It is 10 by default. The dataset we have is not significantly large and using minimum 10 observations in a node will increase bias. Therefore, we tried different values of n.minobsinnod for tuning.

Hyperparameter tuning and Cross validation:

At first, we tried different values of **n.trees** on D.train without cross validation and kept other 3 hyperparameters constant (**d** = **6**, **λ** = **0.1**, **n.minobsinnod** = **5**). We obtained the set of hyperparameters that gives the lowest test MSE of 0.1232. After finding optimal value of **n.trees (= 4400**) through this method, we tried different values of interaction depth keeping **n.trees** = **4400** and other parameters as they are. Our optimum interaction depth was **6** in this case and test **MSE** = **0.1273.** (Diagrams on next page)

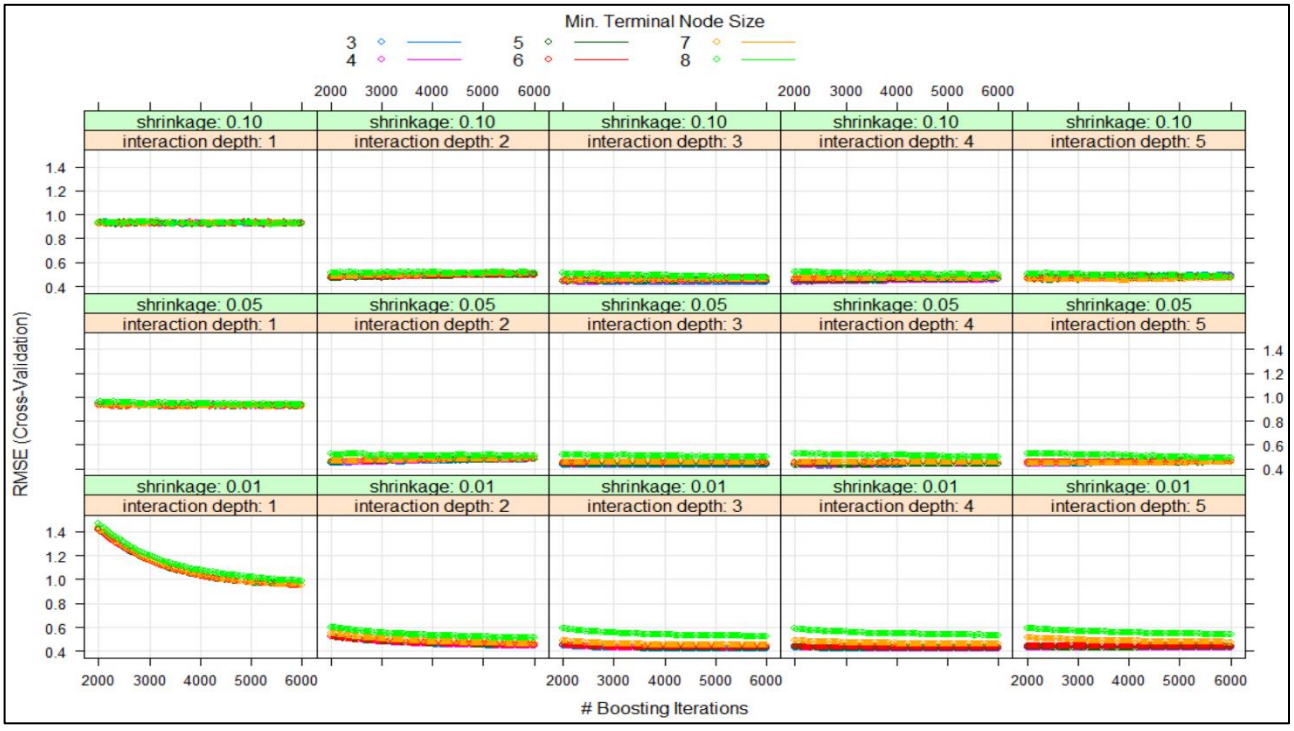




As we kept repeating this procedure, we understood that it is providing unstable results as we are not using any cross-validation methods. Moreover, ideally, we should use optimum grid search method which can parallelly compute test MSE for different combinations of (**n.trees, n.minobsinnod, interaction.depth, shrinkage.**) The reason for this is there are also trade-offs between the hyperparameters, and the previous method doesn't capture those. Therefore, we used caret library for tuning.

Arguments for tuning:

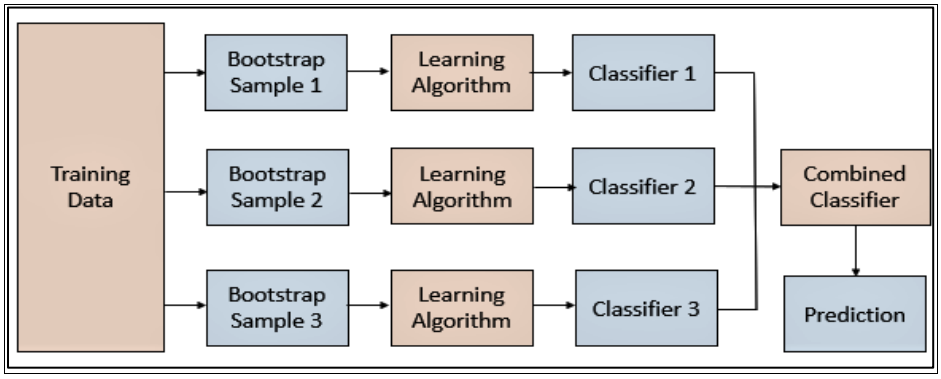
Arguments for tuning	Results and optimum model hyperparameters
GRIDS: • [**interaction.depth** = c(1,2,3,4,5)] • [**n.trees** = seq(2000, 6000, by=100)] • [**shrinkage** = c(0.01, 0.1,0.05)] • [**n.minobsinnode** = c(3,4,5,6,7,8)] ***Training control method*** = 5-Fold cross validation	***Optimum values for hyper parameters:*** [Interaction.depth = **4**], [n.trees = **4850**], [Shrinkage = **0.01**], [n.minobsinnod = **3**] **TEST_MSE** = **0.125** (Validation test) **Platform**: i7 7th gen CPU ram 12 GB, GPU 6 GB We are getting different results for different platforms



Bagging

Overview:

Bagging is also called Bootstrap aggregating. It involves creating multiple samples of the original training data set using the bootstrap technique, applying a learning algorithm, and fitting a separate decision tree on samples selected, and then aggregating all the trees to create a single predictive model. By model averaging, bagging helps to reduce variance and minimize overfitting.



Feature Engineering:

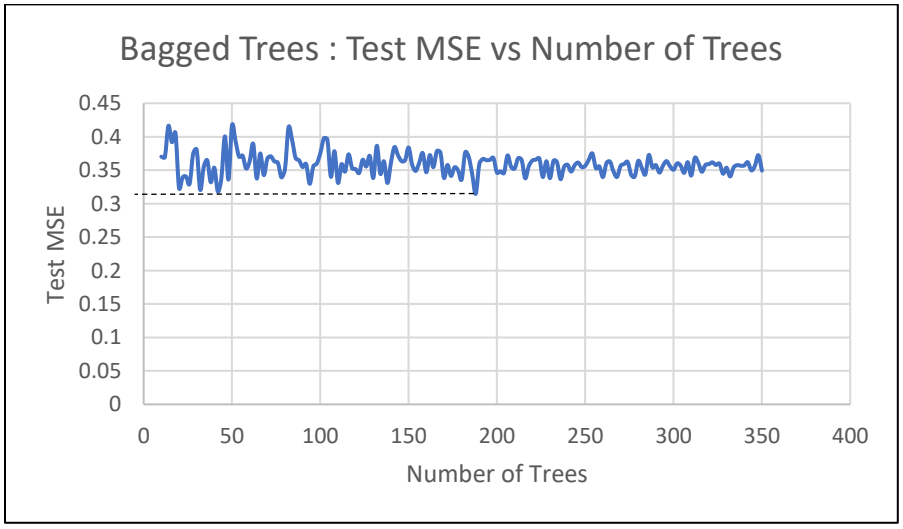
To improve the performance of the built model, we removed insignificant predictors and computed the test MSE. Even after removing the predictors, performance remained the same. Hence, all variables are considered in the final model.

Evaluation Metrics:

To compare bagging with other methods, we used test MSE.

Hyperparameters:

We divide the model in train and test data set to estimate the model performance and accuracy. After splitting the data into two different sets a model was fitted on training data set and the prediction was performed on test data set. While fitting the training model, parameter tuning is one of the important steps. Bagging can be tuned on several parameters and the most important parameters amongst them are ***mtry*** and ***ntree***.



Bagging is a special case of random forest with **m** = number of predictors. Hence, Choose ***mtry*** = ***8***.
The second parameter to be tuned after ***mtry*** is ***ntree***. Grow 10 to 200 number of trees to ensure that every input row gets predicted at least a few times. The results of test MSE vs number of trees are plotted in the below graph. The least prediction error rate was obtained at ***188***th entry. Therefore,

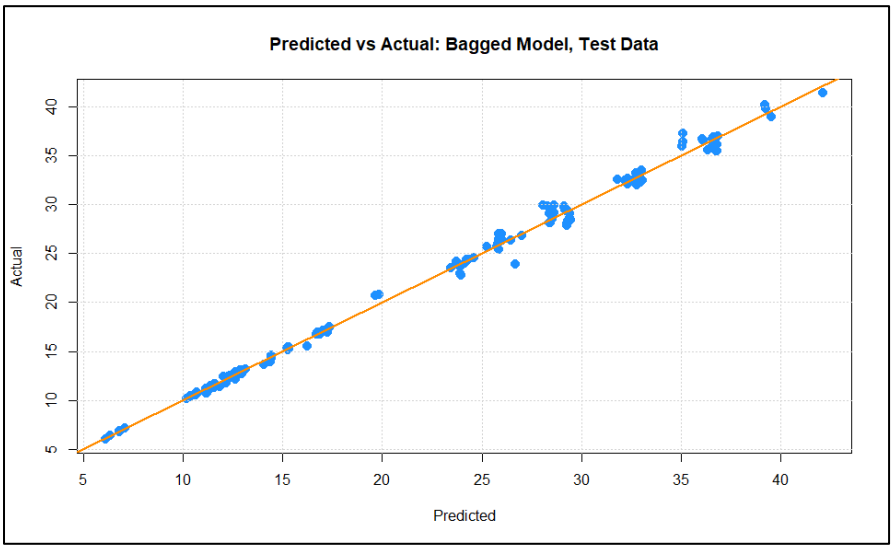
Approximation of Test Error:

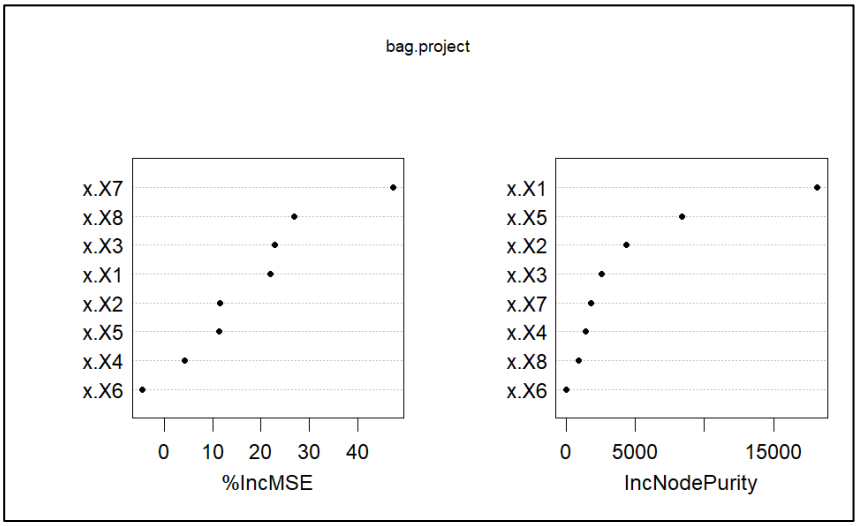
The training MSE associated with the training data set using the final model is **0.1168**.

Bagging uses bootstrapped subsets (i.e. drawing with replacement of the original data set) of training data to generate such an ensemble. Thus, there is no need for separate cross-validation. Using the following combination of tuning parameters, the test error is obtained as:

Tuning Parameters	Test MSE
Mtry = 8, ntree = 188	0.3463

Another way to check the model evaluation is by plotting the predicted vs actual model. The graph is as shown below:
From the graph, we can say that bagging performs well since the line representing predicted data points is in proximity to the original data points.





Predictors which are important from model perspective are plotted in the graph shown on the left-hand side.
For our model predictors x7 and x8 are the most important ones.

Multi-linear regression

Overview:

Multiple Linear Regression primarily models linear relationship between the response and multiple predictors. It can very accurately map the data if the boundary desired is linear, rather than the more complex methods. Although we are modelling a multiple linear regression, we improve and build upon it by changing the flexibility and by introducing interactions between the predictors. Thus, we go through multiple models, and the final model is derived with the help of these initial models.

Data Preparation and Evaluation Matrix:

To determine the quality of fit for a model, we divide the data into training data (70% of the data) and test data (remaining 30% of the data). Then, we apply 10-fold Cross validation method on the training data and compute the Test MSE on the remaining data, i.e., test data. Test MSE is the deciding parameter for comparison between all the models and the model with lowest Test MSE will be selected as the best model.

Technical Analysis:

1. First, we map the Multiple Linear Regression with all the variables.

$$Y_1 = \beta_0 + \beta_1X_1 + \beta_2X_2 + \beta_3X_3 + \beta_4X_4 + \beta_5X_5 + \beta_6X_6 + \beta_7X_7 + \beta_8X_8$$

Below are the metrics we get from the above model.

Adjusted R^2	**90.62%**
Test MSE	**8.277**

Also, by observing the summary, we find that X_4 and X_6 are not significant in predicting the response.

```
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  87.48380   27.65910   3.163 0.001689 **
X1          -69.42042   14.90161  -4.659 4.42e-06 ***
X2           -0.08595    0.02480  -3.467 0.000588 ***
X3            0.05548    0.00951   5.834 1.16e-08 ***
X4                 NA         NA      NA       NA
X5            4.29831    0.48541   8.855  < 2e-16 ***
X6           -0.14584    0.13865  -1.052 0.293511
X7           23.88979    1.63027  14.654  < 2e-16 ***
X8            0.26684    0.09961   2.679 0.007712 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2. Thus, we remove the predictors, X_4 and X_6 and try to fit the linear model with remaining predictors. Below are the metrics we get for the model.

Adjusted R^2	**90.62%**
Test MSE	**8.1618**

Although, the adjusted R^2 remains the same, Test MSE reduced marginally.

3. Next, we try to go for polynomials with higher degree (taking 2 in this case) of the significant predictors to study the effect of increase in flexibility on the model.

Below are the metrics we get for the model.

Adjusted R^2	**90.62%**
Test MSE	**8.1617**

We get almost same results by introducing the quadratic terms, and hence we discard the idea of increasing the flexibility.

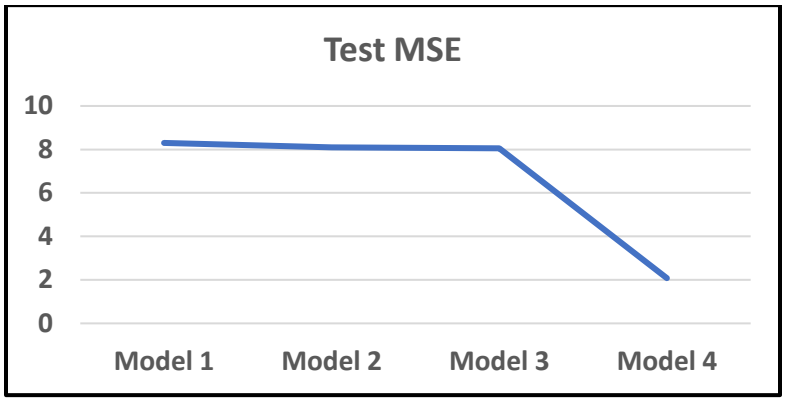
4. At the end, we introduce interactions between the significant predictors. In this model, we introduce all the 2-factor interactions, i.e. ${}^6C_2$ = **15 interactions.**

$$Y_1 = \beta_0 + \beta_1X_1 + \beta_2X_2 + \beta_3X_3 + \beta_5X_5 + \beta_7X_7 + \beta_8X_8 + \beta_9X_1X_8 + \beta_{10}X_2X_8 + \beta_{11}X_3X_8 + \beta_{12}X_5X_8 + \beta_{13}X_7X_8 + \beta_{14}X_1X_7 + \beta_{15}X_2X_7 + \beta_{16}X_3X_7 + \beta_{17}X_5X_7 + \beta_{18}X_1X_5 + \beta_{19}X_2X_5 + \beta_{20}X_3X_5 + \beta_{21}X_1X_2 + \beta_{22}X_1X_3 + \beta_{23}X_2X_3$$

Below are the metrics we get for the model.

Adjusted R^2	**98.01%**
Training MSE	**1.8471**
Test MSE	**2.0870**

We obtain adjusted R^2 value of 98.01% and Test MSE as 2.0870, which is a huge improvement from our previous models. We tried to include more interactions, but the results were not significantly improved. Below is the plot for Test MSE on all the above models. Among all the multi linear regression models, model 4 shows the lowest test MSE.



Comparison between MLR, Bagging and Boosting

Comparative Analysis:

Sr. No.	Model	Errors for best models	
		Train Error	Test Error
1	Boosting	**0.058**	**0.125**
2	Random Forest- Bagging	**0.116**	**0.346**
3	Multi-linear Regression (MLR)	**1.847**	**2.087**

We calculated train and test MSE for each model. For regression, test MSE is a reliable metric that measures the quality of fit after taking into consideration overfitting and underfitting. In case of multi-linear regression and boosting, we have considered test MSE after performing cross validation. In case of bagging, we are already using bootstrapping resampling methods.

Theoretical Approaches for Model Selection:

Boosting v/s Bagging:

The main difference between these two methods is that boosting involves sequentially growing a large tree (learning slowly from many weak trees) and bagging averages predictions from many trees grown on resampled data.

Bias/Variance

Because of relatively low variance in bagging, it is not able to handle the complexity in data. On the other hand, boosting has enough variance to capture complexities in data. We can handle interactions between predictors by varying the interaction depth. Shrinkage is also one of the predictors to control bias-variance trade off in the case of boosting.

Boosting v/s Multi-linear Regression:

Bias/Variance

Being a parametric method, linear model always assumes a specific distribution of data which is restrictive for good prediction. In case of boosting, we do not have such parameters. Rather, we have hyperparameters which are used to find the best fitting model. Boosting handles interactions much better than multi-linear regression.

For example, in MLR, predictor X4 was rendered insignificant but, for boosting, it has high relative importance for prediction.

Conclusion:

Our goal here is to find the best model with the most accurate predictions for unseen data. If our objective were to interpret how the model works, we could have selected linear models despite its high test MSE. Therefore, based on our comparative analyses and findings of the model fitting exercise, we conclude **tree boosting** to be our best model.

Evaluation of Best Model on Test Data

We chose Tree Boosting as our best model based on the performance on validation data set. The hyper-parameters of the model were then fine tuned to obtain the optimal values. These values are shown below:

Hyper-parameters	**Optimal Values**
Interaction.depth	**4**
n.trees	**4850**
Shrinkage	**0.01**
n.minobsinnod	**3**

Our approximation of test mean squared error (MSE) is ***0.125*** (On validation data). After receiving the test data, we generated response predictions for the test data using the optimal tree boosting model. We then computed MSE based on the predicted response values test data. We found that the actual test MSE is ***0.349***

Test MSE Type	**Error**
TEST MSE (Validation Set)	**0.1250**
TEST MSE (Data received on 24th April)	**0.349**

There are several potential reasons why our approximation slightly underestimated the test MSE, including the difference in training and test data in terms of characteristics.

Model Improvement

Overview:

Initially, we used **caret** package for tuning of hyperparameters in our best model (Tree Boosting) using only 70% of the data points. Here, we are reporting the improved test error as a result of hyper-parameter tuning on entire train data.

Data:

Here, we do not split the original training data into train and test sets. We are using the test data (provided on 24th April) as our test set and training our model on all 550 observations. Therefore, we have 218 test observations.

Motivation for improvement:

The test error we obtained on unseen data is **0.349** which is higher than **0.125**, our validation set error. We can now use more training data. We already know that the caret package tunes 4 hyperparameters: **n.trees, interaction.depth, shrinkage,** and **n.minobsinnod.**

More improvement can be done in the model by fine tuning the hyper-parameters on entire training data set. Upon fine tuning these hyper-parameters, the test error was reduced by **23.72%** than that of test error obtained from validation data

Evaluation Metrics:

To compare our improved model with the originally proposed model, we used test MSE.

Hyperparameter tuning and Cross validation:

We again used caret package but this time with all 550 observations as training dataset. We observed that by keeping bag.fraction = 0.5 (default value) we can improve the test MSE.

Here are the optimum values of hyperparameters.

Arguments for tuning	**Results and optimum model hyperparameters**
GRIDS: • [**interaction.depth** = c(1,2,3,4,5)] • [**n.trees** = seq(2000, 6000, by=50)] • [**shrinkage** = c(0.01, 0.1,0.05)] • [**n.minobsinnode** = c(3,4,5,6,7,8)] ***Training control method:*** 5-Fold cross validation	***Optimum values for hyper parameters:*** [Interaction.depth = **4**], [n.trees = **3050**], [Shrinkage = **0.05**], [n.minobsinnod = **3**] ***Improved TEST_MSE*** = **0.266** ***Platform***: i7 7th gen CPU Ram 12 GB, GPU 6 GB We are getting different results for different platforms.