CS6220 - Final Project
Predict the time to become elite user using Yelp's user data set

1. AUTHORS:

The group Id - **G6**

2. INTRODUCTION

The Yelp Dataset has huge set of user, business and review data publicly available for learning purpose. We have analyzed **User data** of Yelp for our project work of Data Mining course. User dataset provides data on the user's profile statistics including a wide variety of information about the spread of other review counts that the user had written previously. It includes some important features such as average stars, total number of reviews, time since joining yelp, elite membership status, year the user became elite, number of Yelp friends, number of fans, details about votes and compliments etc.

2.1 QUESTION TO BE ANSWERED

In this work, we will focus on predicting how long it takes for an active user to become an elite user of Yelp, so that he/she can start enjoying the special privileges of an elite user (which includes free tickets to movies, concert, various discounts etc.). We used information such as review counts, friends count, number of fans, compliments received, votes etc. to estimate the approximate time to become an elite member.

2.2 BENEFITS OF THIS ANALYSIS

Both Yelp and Yelp users can benefit from our prediction. Yelp users can get an idea of how long it would take for them to start receiving special privileges based on the user's current activity. When more people try to become elite, there would be more useful reviews from the users, and this could eventually drive Yelp to be the prime source of reviews on places, businesses etc.

2.3 NON-TECHNICAL SUMMARY

From the information we have, we initially calculate the time taken to become elite. This is done by using information such as yelping since and the first year the user became elite. We then try to predict the time taken to be elite using the information that we have. As we perform the methods that are available to us, we find that the time taken for a user to become elite is mainly dependent on the number of compliments that the user has received. Information such as number of friends, number of useful votes etc. also contribute to our prediction.

3. METHODS

3.1 ANALYSIS ON CORRELATION PLOTS

After cleaning and formatting the data, we began our analysis by creating a correlations plot on the dataset. The correlation plot revealed that the dataset contained many predictors that are highly correlated. This may decrease the prediction accuracy. Hence, in order to eliminate the multicollinearity, we dropped the redundant variables from the dataset. [*Refer appendix 2, 5*]

3.2 NON-LINEARITY OF DATA

Next, we analyzed the residual plots of the linear regression model on the data and found that the relationship between the predictors and response variables are not linear. This lowers the prediction accuracy of the model significantly. Hence, we performed the logarithmic transformation on the predictors to eliminate non-linearity. [*Refer appendix 4*]

3.3 REGRESSION TREES

Since the mean squared error was very high with the linear regression model, we switched gears and created a regression tree on the dataset. Mathematically, the regression tree can be represented as:

$$f(X) = \sum_{m=1}^{M} c_m . 1_{(X \in Rm)}$$

Where f(X) is the predicted time to elite

M is the number of partitions

R_1, R_2, …… R_M represents partitions of feature space

First we created a subset of elite users from the full dataset [*Refer appendix 1*]. We then divided the subset into two equal sizes for training and validation [*Refer appendix 7*]. Next, we created a tree using the training data and validated the tree on the validation set using cross validation technique. The resulting MSE was much lower when compared to the linear regression model [*Refer appendix 8, 9, Table 1*].

We went on to experiment with other methods to improve the performance of the model.

3.4 BAGGING

Since the regression tree suffers from high variance, we can reduce the variance using bootstrap aggregation or bagging. Bagging is a special case of random forest, where we set the number of variables randomly sampled as candidates at each split (m) equal to the number of predictors in the dataset (p).

Mathematically, bagging can be represented as:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

Where B is the number of bootstrapped training datasets.

When we applied bagging to our dataset, we saw very little improvement in the MSE over the regression tree. [*Refer appendix 10, Table 1*]

3.5 RANDOM FOREST

Since selecting the predictor subset size to be equal to the number of predictors in the dataset did not make any significant improvement in predicting the time to become an elite member, we chose to tweak the predictor subset size as m = p/3 in each split, making the average of the resulting trees less variable and hence more reliable.

Performing random forest on the training dataset and validating it using the validation set resulted to be fruitful. The MSE decreased by a significant value when compared to the mean squared errors obtained from bagging and regression tree. [*Refer appendix 11, Table 1*]

3.6 BOOSTING

Boosting works in a similar way to bagging, except that the trees are grown sequentially. Boosting process can be mathematically shown as:

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^{b}(x).$$

Where λ is the shrinkage factor which slows the process down even further, allowing more and different shaped trees to attack the residuals. When we applied boosting to our elite dataset using number of trees (n) = 5000, we could see that the mean squared error rate dropped further when compared to the other approaches that we used.
[*Refer appendix 12, 13, 14, Table 1*]

4. RESULTS

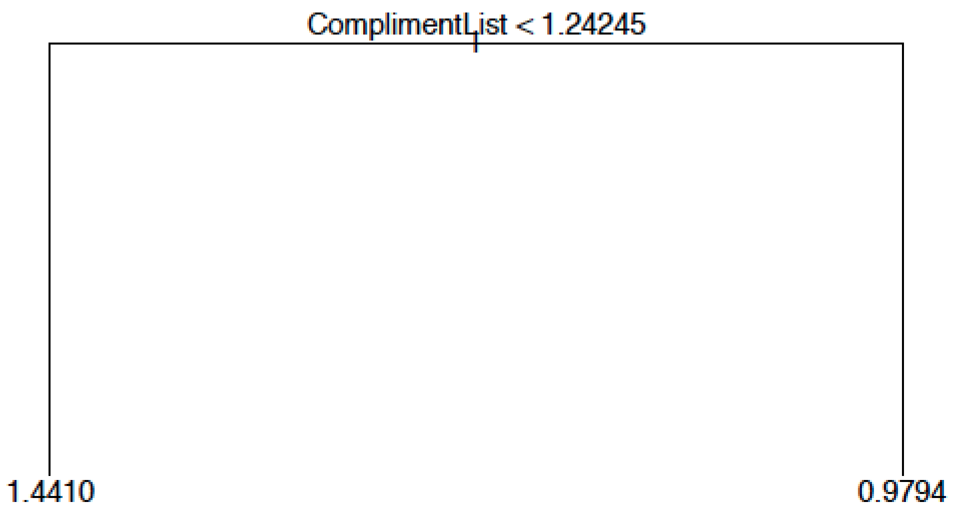


Fig 1. Regression Tree Output

The results obtained are based on the analysis of the dataset using Regression Tree approach. The Regression tree method predicted the time to become an elite member (i.e. TimeToElite), by primarily using 1 predictor, ***ComplimentList*** from the elite dataset.

The tree that is formed, has 1 internal node and 2 terminal nodes. The split happened on the basis of ComplimentList. If ComplimentList < 1.24245, then the predicted TimeToElite is 1.4410 years, else if ComplimentList >= 1.24245, then it takes a user 0.9794 years to become an elite user.

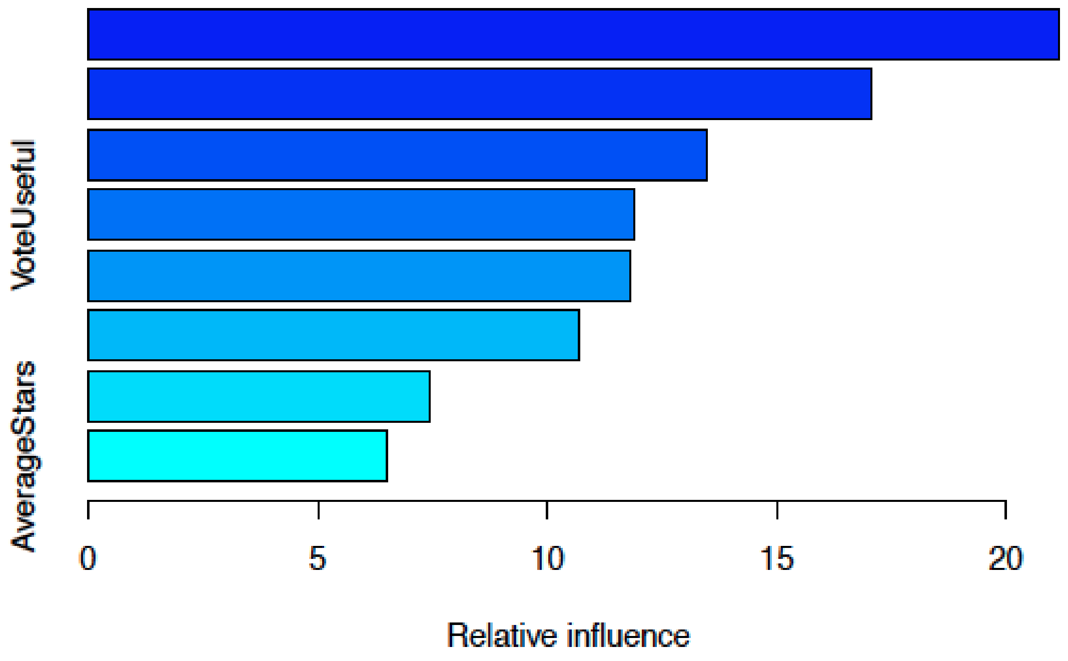
METHODS	**MSE**	$\sqrt{MSE}$
Regression tree	1.38647	1.177485
Bagging	1.385713	1.177163
Random Forest	1.359728	1.166074
Boosting	1.332325	1.154264

Table 1 - Summary of MSE & $\sqrt{MSE}$ of each approach

On applying Bagging, Random Forest and Boosting, to improve the prediction accuracy we see that the mean squared error rate obtained follows the below trend.

Bagging > **Random Forest** > **Boosting**

The predicted TimeToElite obtained on performing bagging, random forest and boosting lie within the range of 1.177, 1.166 and 1.154 years respectively from their actual values. On analyzing the summary of boosting, we get a relative influence plot and the relative influence statistics as shown below:



```
##                               var   rel.inf
## ComplimentList     ComplimentList 21.141767
## FriendsCount         FriendsCount 17.041596
## ComplimentHot       ComplimentHot 13.482239
## VoteUseful             VoteUseful 11.896983
## ComplimentPhotos ComplimentPhotos 11.798063
## ReviewCount           ReviewCount 10.694879
## Fans                         Fans  7.429253
## AverageStars         AverageStars  6.515218
```

5. DISCUSSION

The learning has been immense. We came across quite a few surprises:

The most prominent one being, the predictors that were actually used for the model. On taking a first look at the dataset, we expected ReviewCount, Fans and AverageStars to be the most important predictors. But after performing Regression trees, we found that the predicted TimeToElite is not very dependent on any of the above predictors. [*Refer appendix 13, 14*]

Another point that stood out, was the number of predictors that is used in predicting the model. Considering the size of the dataset, which has around 22 predictors, we expected the model to contain quite a few internal nodes. Instead, the regression tree considered only one internal node, which is ComplimentList, and the split was performed on that.

Potential Steps to Improve the Analysis:

1) Varying shrinkage value (λ) while performing boosting could help get a lower value of MSE, which in turn could lead to a more accurate model.
2) Creating a synthetic variable that better explains the behavior of the highly correlated group of variables.

6. REFERENCES:

1. Yelp Dataset Challenge - http://www.yelp.com/dataset_challenge
2. An Introduction to Statistical Learning with Applications in R *by* Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani
3. Inside-R - http://www.inside-r.org
4. Stack Overflow - http://www.stackoverflow.com

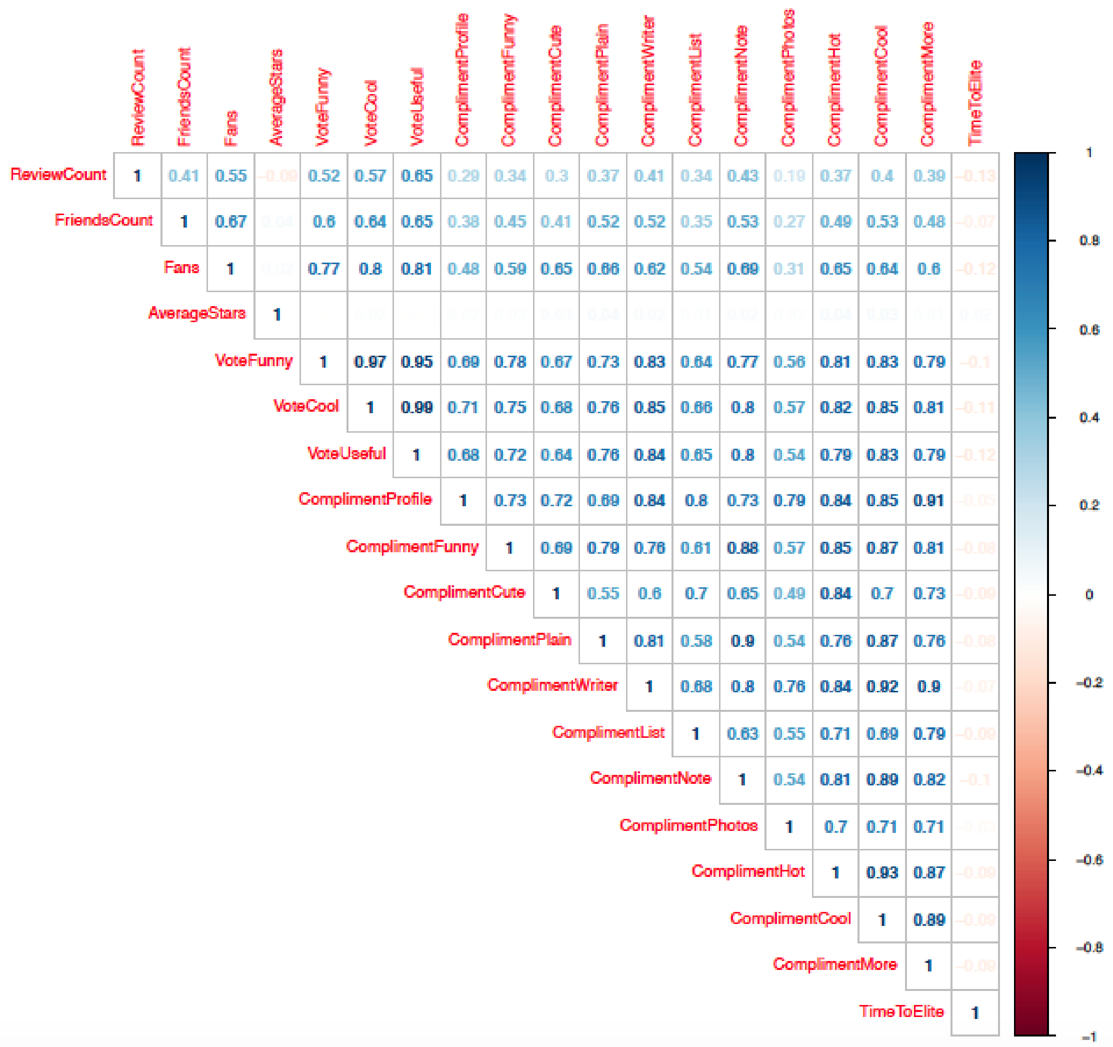
APPENDIX

Appendix 1: Create subset, *yelpElite,* of Elite members

```
# Remove non-Elite members from dataset.
yelpElite = yelp[(yelp$EliteYearCount > 0), ]
```

Appendix 2: Obtain the correlation plot between all predictors

```
library('corrplot')
corrplot(cor(yelpElite), # Remove non-numeric columns
         method = "number", type = "upper")
```

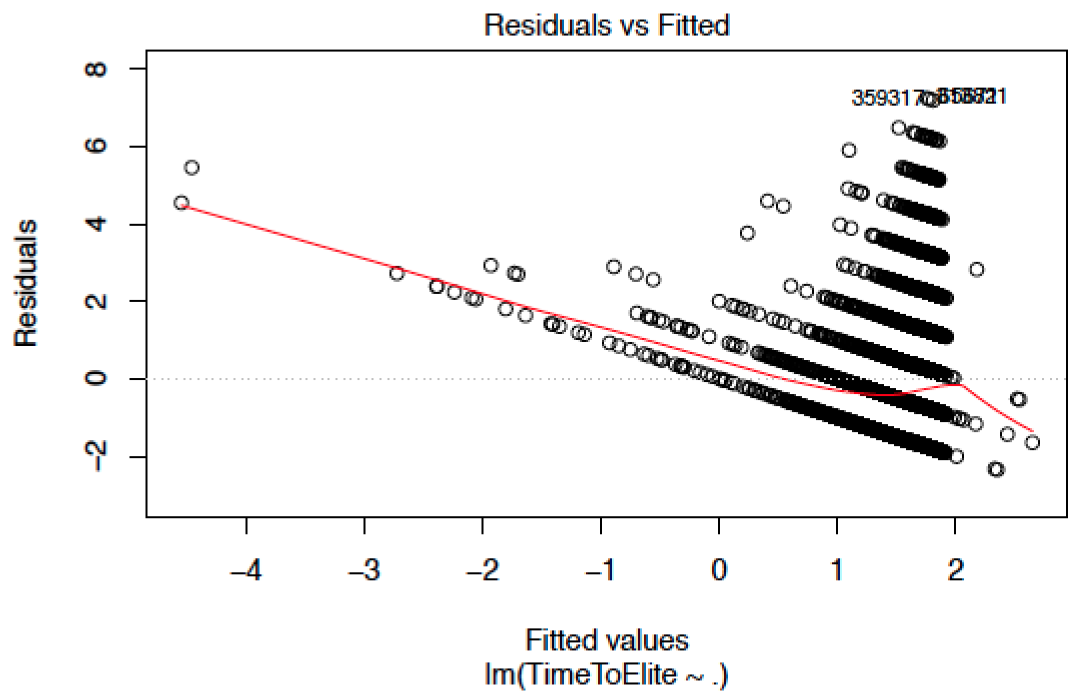


	ReviewCount	FriendsCount	Fans	AverageStars	VoteFunny	VoteCool	VoteUseful	ComplimentProfile	ComplimentFunny	ComplimentCute	ComplimentPlain	ComplimentWriter	ComplimentList	ComplimentNote	ComplimentPhotos	ComplimentHot	ComplimentCool	ComplimentMore	TimeToElite
ReviewCount	1	0.41	0.55	−0.09	0.52	0.57	0.65	0.29	0.34	0.3	0.37	0.41	0.34	0.43	0.19	0.37	0.4	0.39	−0.13
FriendsCount		1	0.67	0.04	0.6	0.64	0.65	0.38	0.45	0.41	0.52	0.52	0.35	0.53	0.27	0.49	0.53	0.48	−0.07
Fans			1	0.02	0.77	0.8	0.81	0.48	0.59	0.65	0.66	0.62	0.54	0.69	0.31	0.65	0.64	0.6	−0.12
AverageStars				1	[illegible]	0.02	[illegible]	0.02	0.02	0.03	0.04	0.02	0.01	0.02	0.02	0.04	0.03	0.01	0.02
VoteFunny					1	0.97	0.95	0.69	0.78	0.67	0.73	0.83	0.64	0.77	0.56	0.81	0.83	0.79	−0.1
VoteCool						1	0.99	0.71	0.75	0.68	0.76	0.85	0.66	0.8	0.57	0.82	0.85	0.81	−0.11
VoteUseful							1	0.68	0.72	0.64	0.76	0.84	0.65	0.8	0.54	0.79	0.83	0.79	−0.12
ComplimentProfile								1	0.73	0.72	0.69	0.84	0.8	0.73	0.79	0.84	0.85	0.91	−0.05
ComplimentFunny									1	0.69	0.79	0.76	0.61	0.88	0.57	0.85	0.87	0.81	−0.08
ComplimentCute										1	0.55	0.6	0.7	0.65	0.49	0.84	0.7	0.73	−0.09
ComplimentPlain											1	0.81	0.58	0.9	0.54	0.76	0.87	0.76	−0.08
ComplimentWriter												1	0.68	0.8	0.76	0.84	0.92	0.9	−0.07
ComplimentList													1	0.63	0.55	0.71	0.69	0.79	−0.09
ComplimentNote														1	0.54	0.81	0.89	0.82	−0.1
ComplimentPhotos															1	0.7	0.71	0.71	−0.03
ComplimentHot																1	0.93	0.87	−0.09
ComplimentCool																	1	0.89	−0.09
ComplimentMore																		1	−0.09
TimeToElite																			1

Appendix 3: Fit linear model and obtain Residuals vs Fitted Plot

```
# diagonistic plot
lm.yelp <- lm(TimeToElite~., data=yelpElite)
plot(lm.yelp)
```

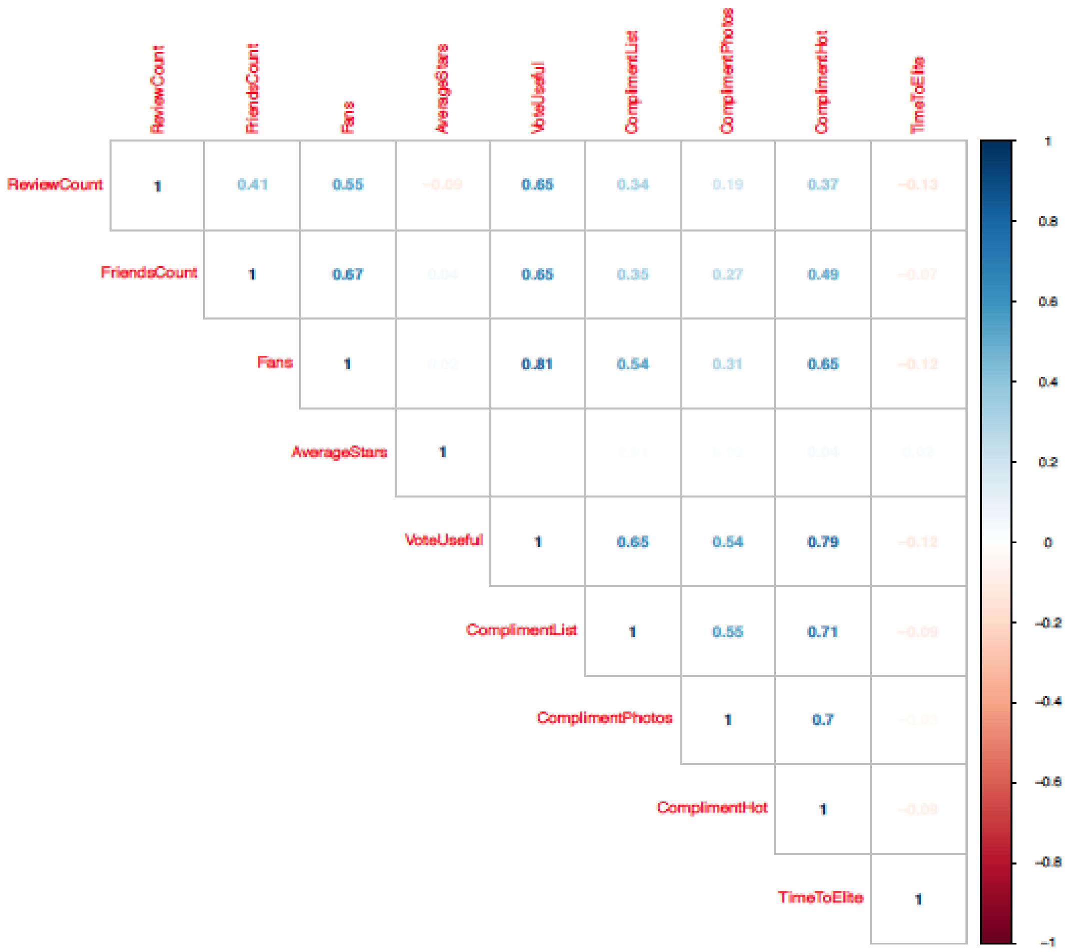
Residuals Vs Fitted Plot with non-linearity



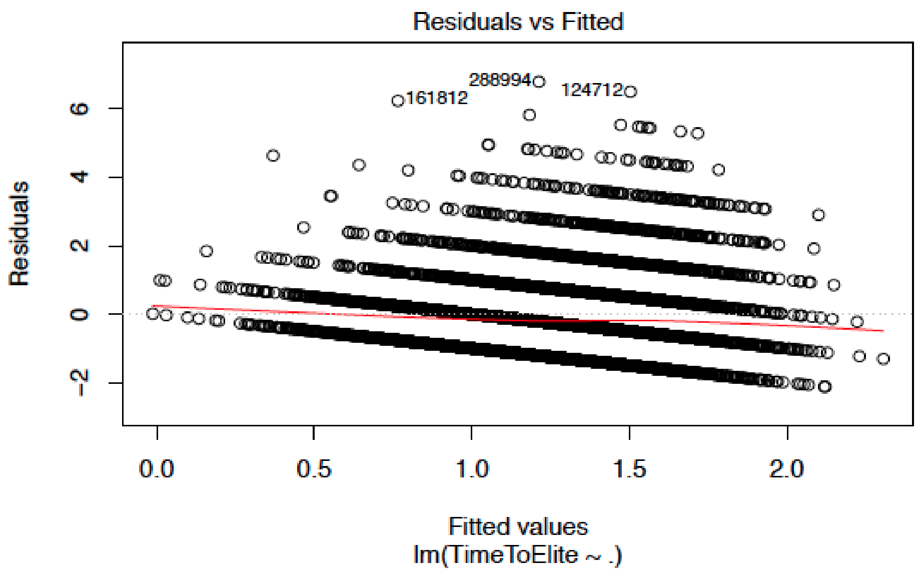
Appendix 4: Convert non-linear values to linear values using log transformation and remove infinite values

```
yelpEliteln$ReviewCount <- log(yelpEliteln$ReviewCount)
yelpEliteln$FriendsCount <- log(yelpEliteln$FriendsCount)
yelpEliteln$Fans <- log(yelpEliteln$Fans)
yelpEliteln$VoteUseful <- log(yelpEliteln$VoteUseful)
yelpEliteln$ComplimentList <- log(yelpEliteln$ComplimentList)
yelpEliteln$ComplimentPhotos <- log(yelpEliteln$ComplimentPhotos)
yelpEliteln$ComplimentHot <- log(yelpEliteln$ComplimentHot)


# Remove infinite

yelpEliteln<- yelpEliteln[is.finite(yelpEliteln$ReviewCount)
                          &is.finite(yelpEliteln$FriendsCount)
                          &is.finite(yelpEliteln$Fans)
                          &is.finite(yelpEliteln$VoteUseful)
                          & is.finite(yelpEliteln$ComplimentList)
                          & is.finite(yelpEliteln$ComplimentHot)
                          & is.finite(yelpEliteln$ComplimentPhotos)
                          , ]
```

Appendix 5: Correlation plot without non-linearity



Appendix 6: Residuals Vs Fitted Plot without non-linearity

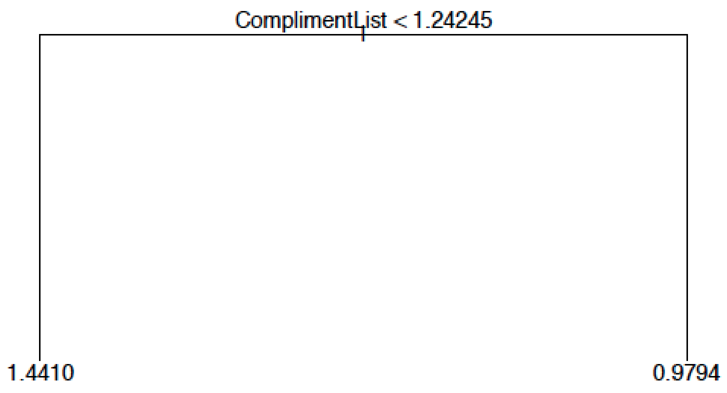


Appendix 7: Divide data equally into training and test set

```
# Now that the data set is clean and linear, We divide it into training and validation
set.seed (1)
trainVector=sample(1: nrow(yelpEliteln), nrow(yelpEliteln)/2)
train = yelpEliteln[trainVector, ]
test= yelpEliteln[-trainVector, ]
```

Appendix 8: Create Regression Tree

```
library(tree)
tree.train = tree(TimeToElite ~ .,  data= train)
plot(tree.train)
text(tree.train ,pretty =0)
```



Appendix 9: Perform test prediction on unpruned tree

```
# PERFORMING TEST PREDICTIONS ON UNPRUNED TREE
yhat=predict(tree.train, newdata=test)
timeToEliteTest.test= test[,"TimeToElite"]
testMSE = mean((yhat-timeToEliteTest.test)^2)
testMSE
sqrt(testMSE)
```

Appendix 10: Perform Bagging & Test Predictions

```
library(randomForest)
set.seed(1)
bag.yelp=randomForest(TimeToElite ~., data = train, mtry=8, importance =TRUE)
# PERFORMING TEST PREDICTIONS ON BAGGED TREE
yhat.bag=predict(bag.yelp, newdata=test)
testMSE = mean((yhat.bag-timeToEliteTest.test)^2)
testMSE
sqrt(testMSE)
```

Appendix 11: Perform Random Forest & Test Predictions

```
# PERFORMING RANDOM FOREST
set.seed(1)
rf.yelp =randomForest(TimeToElite ~., data=train, importance =TRUE)
```

```
# PERFORMING TEST PREDICTIONS ON RANDOM FOREST
yhat.rf = predict(rf.yelp ,newdata=test)
testMSE = mean((yhat.rf - timeToEliteTest.test)^2)
```

```
sqrt(testMSE)
```

Appendix 12: Perform Boosting & Test Predictions

```
library(gbm)
```

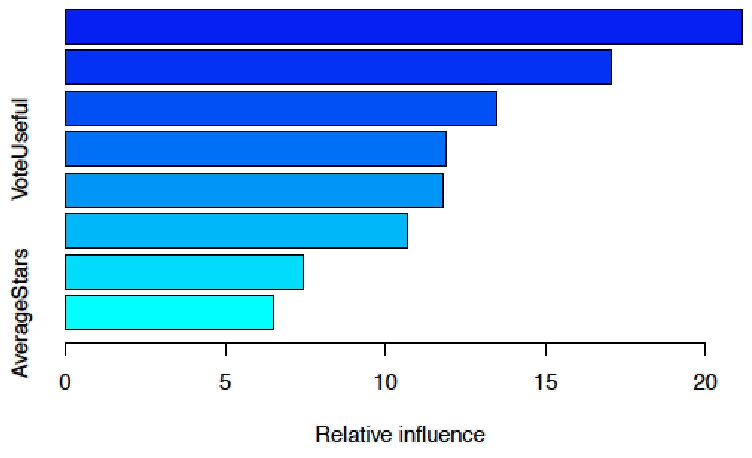
```
## Loading required package: survival
## Loading required package: lattice
## Loading required package: splines
## Loading required package: parallel
## Loaded gbm 2.1.1
```

```
set.seed(1)
boost.yelp=gbm(TimeToElite~.,data = train, distribution="gaussian",
               n.trees =5000, interaction.depth = 4)
# Summary draws the relative influence plot and relative influence statistics
summary(boost.yelp)
```

```
# PERFORMING TEST PREDICTIONS ON BOOSTING
yhat.boost=predict(boost.yelp, newdata=test, n.trees =5000)
testMSE = mean((yhat.boost - timeToEliteTest.test)^2)
testMSE
```

```
sqrt(testMSE)
```

Appendix 13: Boosting Relative Influence Plot

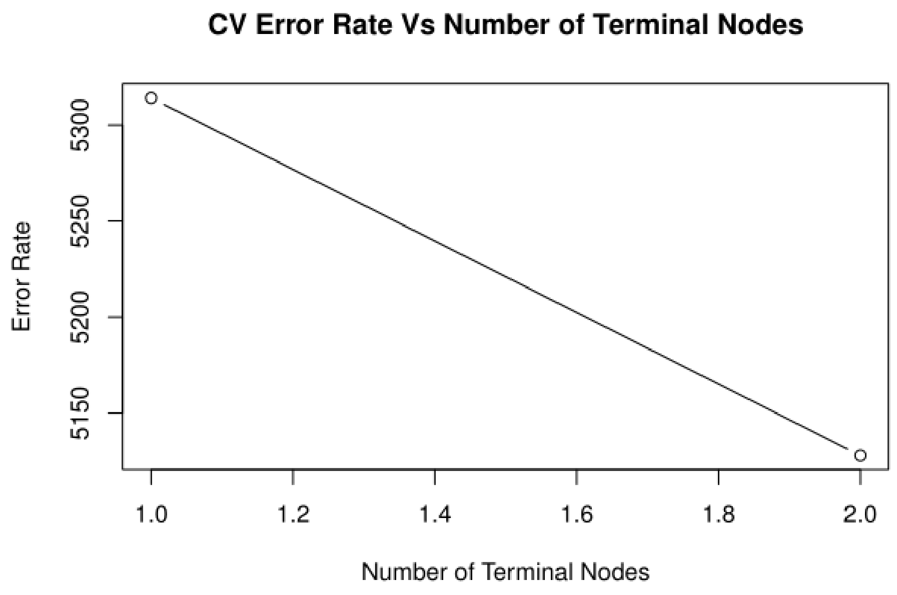


Appendix 14: Boosting Relative Influence Statistics

```
##                                var   rel.inf
## ComplimentList     ComplimentList 21.141767
## FriendsCount         FriendsCount 17.041596
## ComplimentHot       ComplimentHot 13.482239
## VoteUseful             VoteUseful 11.896983
## ComplimentPhotos ComplimentPhotos 11.798063
## ReviewCount           ReviewCount 10.694879
## Fans                         Fans  7.429253
## AverageStars         AverageStars  6.515218
```

Appendix 15: Cross validation Error Rates

```
cv.yelp =cv.tree(tree.train )
plot(cv.yelp$size ,cv.yelp$dev ,type='b',
     xlab = "Number of Terminal Nodes",
     ylab = "Error Rate",
     main = "CV Error Rate Vs Number of Terminal Nodes")
```



STATEMENT OF CONTRIBUTION

Authors:

1. Ashish Khanna
2. Kevin George Abraham
3. Revon Sarah Mathews
4. Sandeep Ramamoorthy

Contributions:

We split the group into 2 sub group -
1. Ashish and Kevin
2. Revon and Sandeep

Dataset conversion (JSON to CSV format):

Sandeep and Revon loaded JSON object to database.
Ashish and Kevin took database backup and generated csv format of the user data.

Data analysis:

The entire team reviewed the csv file manually to understand the data pattern and tried to find some relationship by just reviewing the data.

Data cleansing:

Sandeep and Revon worked on cleaning data (unwanted columns and records with missing values were removed from the dataset).

Collinearity check:

Ashish and Kevin checked the collinearity in data using correlation plots and removed highly correlated predictors to improve the prediction accuracy of the model.

Model analysis:

Sandeep and Revon analyzed the data using Multiple Regression model.
Ashish and Kevin analyzed the data using Regression Tree model.

Validate the analysis:

Sandeep and Revon validated the Regression tree model using the validation set.
Ashish and Kevin validated the Multiple Regression model using the same validation set.

Regression Tree Model:

Entire team worked together on regression tree in more detail as regression tree model seemed to fit our dataset better.

Documentation:

Entire team worked together to produce the final project report.