Analyzing Message Popularity in Group Chats

~ AUTHOR1 and AUTHOR2 ~

Table of Contents

1. **Summary of Research Questions:**
    a. *How can we process messages that makes them machine readable?*

In order to make our message data machine readable, we need to transform the raw text message into numerical. We developed 11 features that defines each message. Based on these set of features (polarity, subjectivity, word count, average word length, adjective ratio, verb ratio, noun ratio, mentions count, urls count, exclamation mark count, question mark count), we converted each message into its corresponding language features. By inputting numeric language features instead of raw messages,our program will be able to process the message data.

b. How can we train a proper model to recognize which features are relevant to popularity?

Under this problem, we first tried to find the setting for most optimized decision tree regressor model regarding message popularity prediction. To do this, we first divided language feature generated data it into training, dev, and testing sets. Then, through a series of loops, we trained hundreds of models with tweaked hyperparameters and tested the model against dev set try to find the minimum test mean squared error, mse. Then, with these hyperparameters, we can create an optimized model and test it against the testing set to figure out how well our model performs. With a relatively high predicting accuracy model, we plotted out the internal decision tree of our model. From the decision tree we can easily find out the most important language features regarding message popularity.

c. How can we understand the effect of each language features on message popularity?

With the most important language features found in the last step, we can groupBy each user's predicted popularity with mean, select out the most and least popular users and compare their respective important features we found above. With these datas, we can reference the model diagram and confirm our model's splits by graphically represent the relationship between most important features and users' popularity. In the end, after analyze the graph we generated, we will be

able to better understand the effect of each important language features on message popularity.

2. Motivation and Background

Our motivation for this project began with an interest in our own group chat. We both participate in a small group chat among a large group of friends through GroupMe. Our original idea stemmed from an idea of attempting to learn the patterns of messages our friends sent in order to attempt to identify which message was sent by which friend through a machine learning algorithm. We then expanded this idea to focus on a broader detail: the linguistic features that differ between a more popular and less popular user within a group chat community. What are the most influential language features on determining high popularity and low popularity users, and how do these features specifically influence the popularity. This research will allow us to find out how to tailor our own messaging style so that a bigger amount of people will more likely to read out message and help us become more influential.

3. Dataset

Our main dataset comes from a Gitter's public chat room between 2014 to 2017. The chatroom is called freecodecamp, which contains around 5 million messages and upwards of 400,000 users. These messages where generously uploaded on kaggle for educational purposes. It is in the format of a csv and contains column including user information, message information along with time, how many people read the message and much.

For more information about the dataset please look up here: https://www.kaggle.com/freecodecamp/all-posts-public-main-chatroom

Our secondary dataset is a small private dataset from the Groupme chats between our friends. We provided an anonymized and serialized version of the data.

4. Methodology

Our goal is to create a machine learning algorithm (more specifically, a decision tree regressor) which predict message's popularity given text message. and then manually use natural language processing techniques to understand our machine learning algorithm.

Step 1:

From our original dataset, we need to clean the data and create a more useful dataset that is only composed of the user data, the message, the message id, the mentions, the URLs and the read bys. Then we will expand the message by doing natural language processing on message datas: First tokenize the message line and get rid of less meaningful part of each message (for example, breaking words). Second, we can create new columns containing language features that our learning algorithm can understand. This includes average word length, sentiments of the sentence, adjective to word ratio, and other features mentioned earlier.

Step 2:

With these features we need to construct a model, a decision tree regressor. We should now split our data with 60:20:20 where the majority data is training the other other is dev and testing. This allows us to tune hyperparameters which help the model perform better. At this point, we need to loop through various hyperparameters and find which hyperparameters lead to the lowest error (mean squared error) with error being calculated by a model predicting against the dev set.

Once we finalize our hyperparameters, we can create a model with all the parameters and test it against the testing set to calculate a score (mse) which

classifies our models accuracy. This model is now ready to predict popularity scores given textual features.

Step 3:

Given our optimized machine learning model, we can now dive deeper into the textual analysis and understand why certain features were prioritized than others.

Use the optimized machine learning model we build earlier to predict the popularity of active users we selected from whole dataset. Then we will manually analyze influential features we find in step 2 to investigate their relationship with each users' predicted popularity. After further understand how do these features truly affect users' popularity and explain the decision of machine learning model, we will be able to answer our research question: how to make our messages more popular in this group chat?

5. Result

a. How can we process messages that makes them machine readable?

In this step, we used three python files, main.py, features.py, clean_data.py to process the raw CSV data we get online.

In the clean_data.py, we wrote a function called clean(), which takes in CSV file, removes data rows with text message as noun, and extract data columns that we will be need for further analysis, which is 'fromUser.displayName', 'fromUser.username', 'fromUser.id', 'mentions', 'urls', 'readBy', 'editedAt', 'id', 'text'. The other function in clean_data.py is clean_sentence(), which uses regular expression to remove any non-alphabetic content of all the message in 'text' column.

In the features.py, we have a pack of assistant method and a main method called get_features() that work together to generate a list of features, which includes polarity, subjectivity, word count, average word length, adjective ratio,

verb ratio, noun ratio, mentions count, urls count, exclamation mark count, question mark coun.

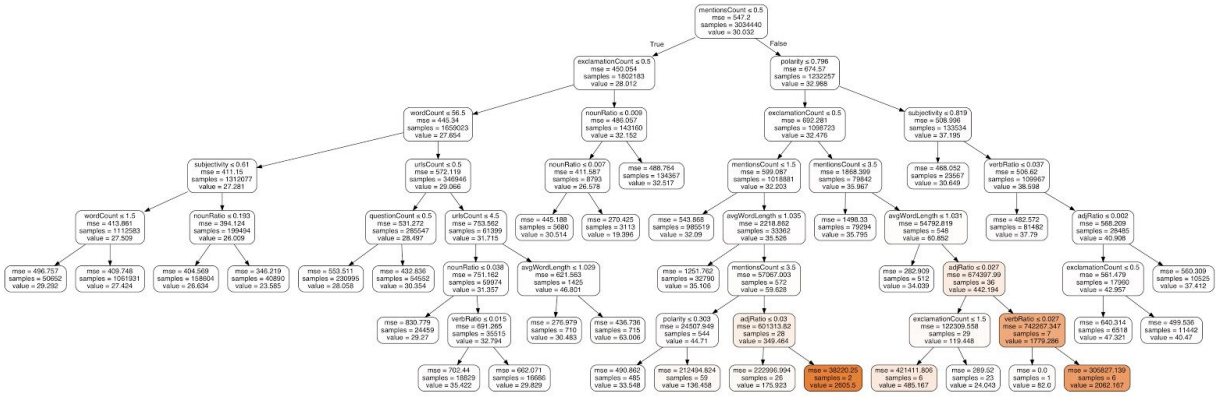
In main.py, we imported both clean_data.py and features.py. Taking the CSV file as input, main.py read in original dataframe, clean it up, get all the features we want from the 'text' column and generate a new dataframe contains selected columns and new feature columns. Because we converted raw text message into columns of numeric features, this new Dataframe will be easily processed by python programs. At the end, we pickled the DataFrame into data.pkl for later use.

b. How can we find out the most influential language features regarding messages' popularity?

In this part, we spent most of our efforts testing our machine learning model trying to figure out the most influential hyperparameter and their most optimized setting for our machine learning model that can give back the highest prediction accuracy.

After our testing, we find out *mac_depth* and *max_leaf_nodes* are the two most influential parameters regarding our model's prediction accuracy. Eventually we decided to set out model as *max_depth*=6, *max_leaf_nodes*=31, which according to multiple testing has the highest prediction accuracy.

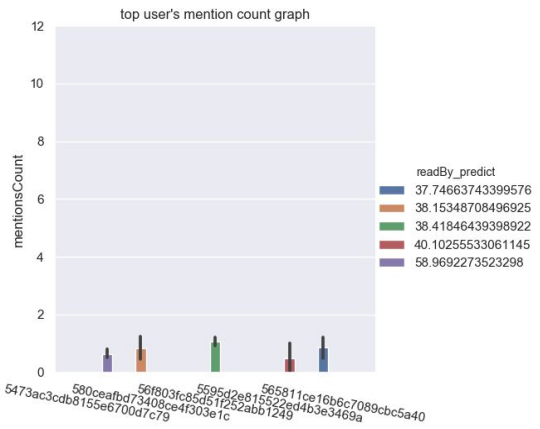
In the end, we used the used our previously prepared testing data to train this mode, and printed out the decision tree graph inside of our model:

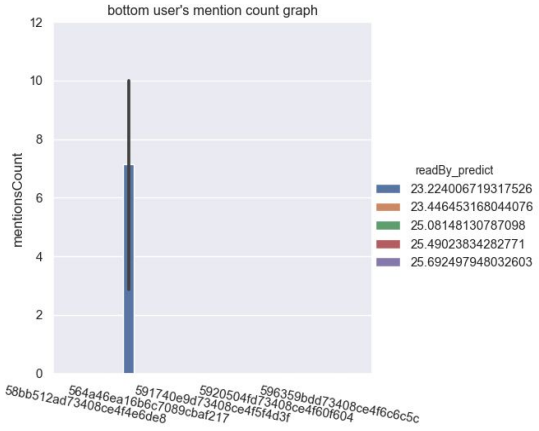


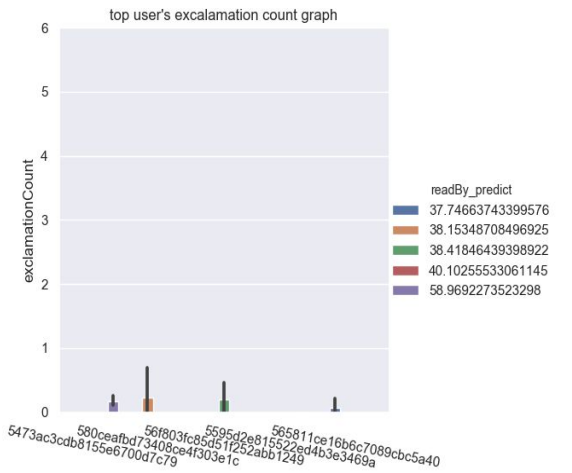
From a brief analysis of the decision tree model shows above, we can clearly tell that ***mention count, exclamation ratio, polarity, word count, noun ratio, and subjectivity*** were the four most influential language features in this machine learning model regarding the prediction of a message's popularity.

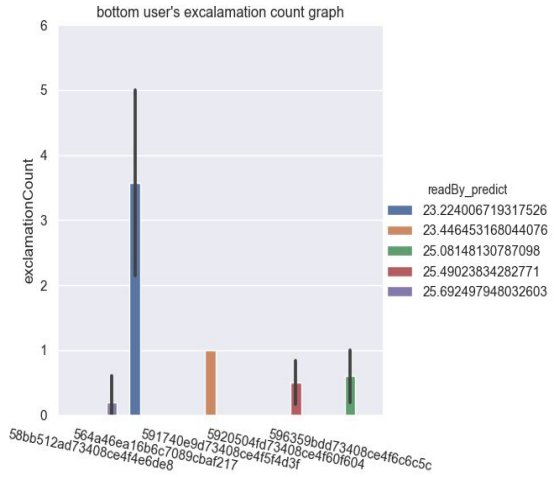
c. How can we understand the effect of each language features on message popularity?

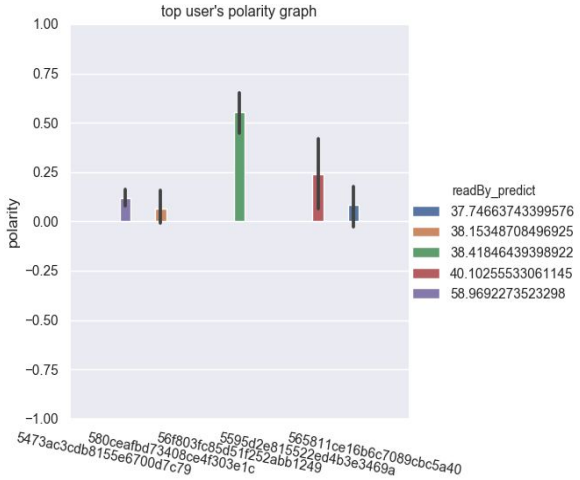
In order to further understand our machine learning model, we used the previously built machine learning model to predict a readBy number for every message in our test data, and then calculated the **Average Predicted ReadBy (APR)** for all the users in test data based on their messages' Predicted ReadBy and sorted users in descending order based on average predicted readBy. After filtering out all the low active users(message count < 5), we selected the top 5 and bottom 5 users regarding their APR number. In the end, we used seaborn library's catplot object to plot out the relationship between top 5 users, bottom 5 users and their the tree language features (***mention count, exclamation ratio, polarity, word count, noun ratio, and subjectivity)*** we found out in previous question:

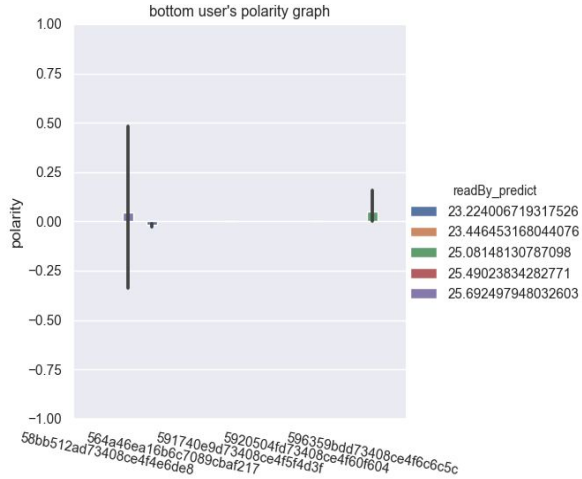


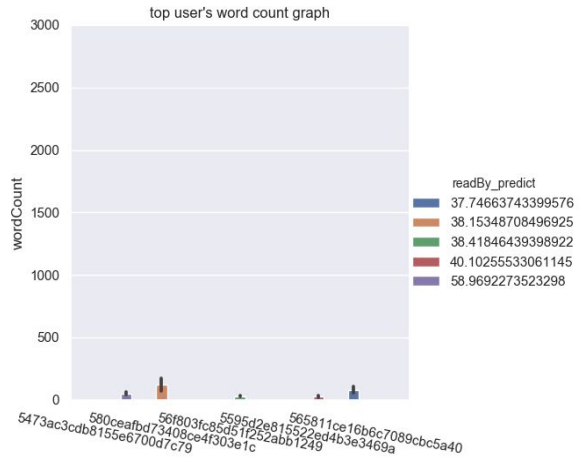


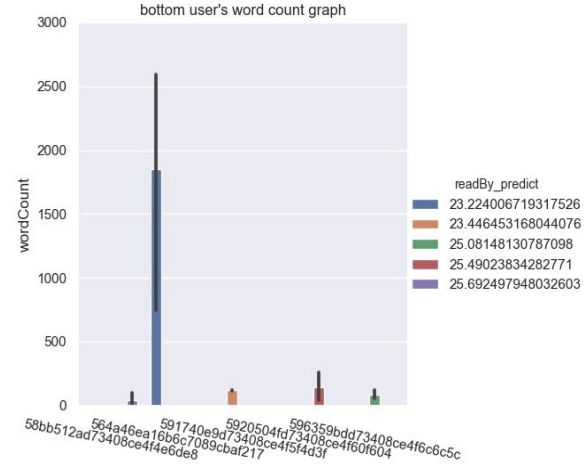

top user's excalamation count graph
exclamationCount
readBy_predict
37.74663743399576
38.15348708496925
38.41846439398922
40.10255533061145
58.9692273523298
5473ac3cdb8155e6700d7c79
580ceafbd73408ce4f303e1c
56f803fc85d51f252abb1249
5595d2e815522ed4b3e3469a
565811ce16b6c7089cbc5a40

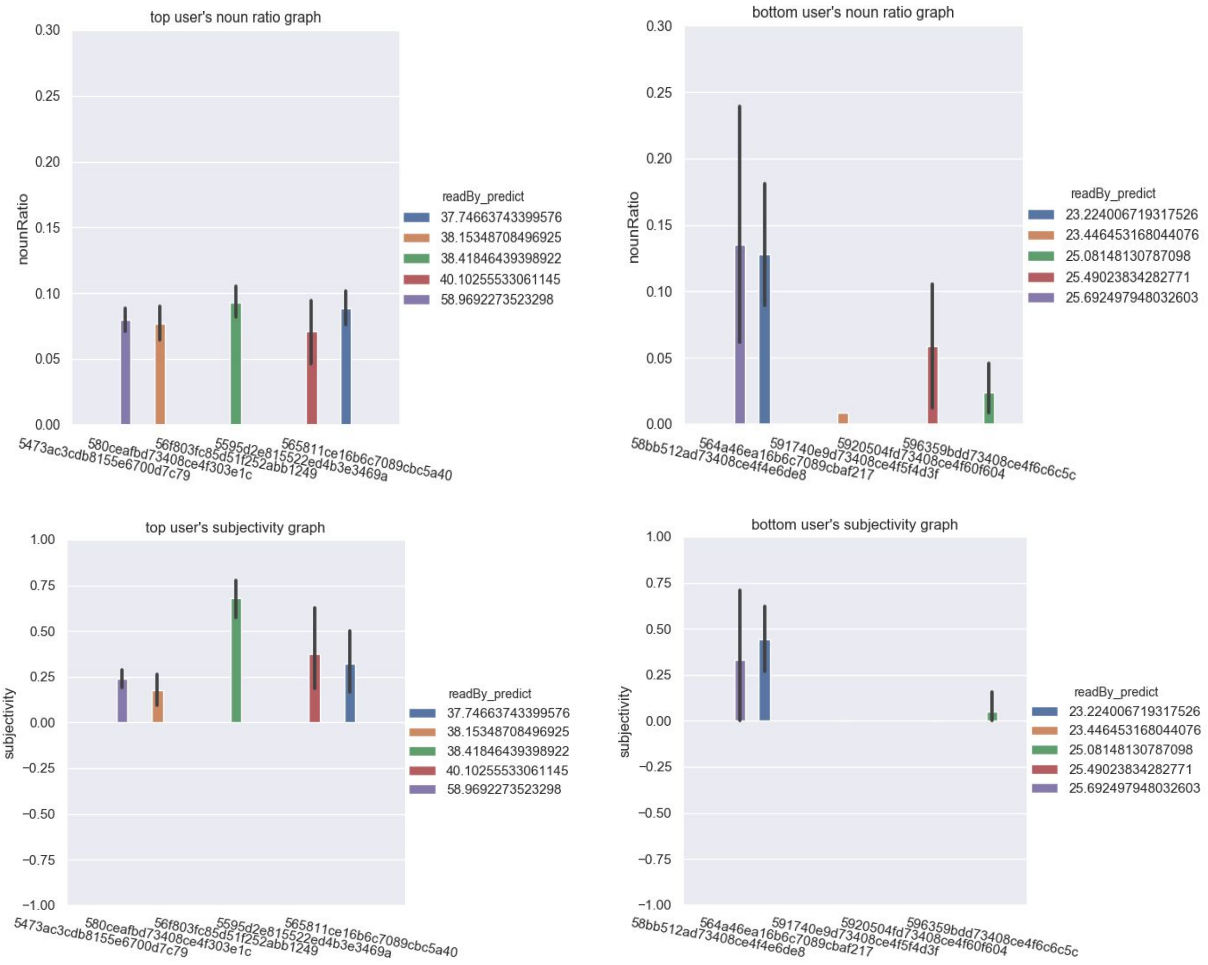

bottom user's excalamation count graph
exclamationCount
readBy_predict
23.224006719317526
23.446453168044076
25.08148130787098
25.49023834282771
25.692497948032603
58bb512ad73408ce4f4e6de8
564a46ea16b6c7089cbaf217
591740e9d73408ce4f5f4d3f
5920504fd73408ce4f60f604
596359bdd73408ce4f6c6c5c


top user's polarity graph
polarity
readBy_predict
37.74663743399576
38.15348708496925
38.41846439398922
40.10255533061145
58.9692273523298
5473ac3cdb8155e6700d7c79
580ceafbd73408ce4f303e1c
56f803fc85d51f252abb1249
5595d2e815522ed4b3e3469a
565811ce16b6c7089cbc5a40


bottom user's polarity graph
polarity
readBy_predict
23.224006719317526
23.446453168044076
25.08148130787098
25.49023834282771
25.692497948032603
58bb512ad73408ce4f4e6de8
564a46ea16b6c7089cbaf217
591740e9d73408ce4f5f4d3f
5920504fd73408ce4f60f604
596359bdd73408ce4f6c6c5c


top user's word count graph
wordCount
readBy_predict
37.74663743399576
38.15348708496925
38.41846439398922
40.10255533061145
58.9692273523298
5473ac3cdb8155e6700d7c79
580ceafbd73408ce4f303e1c
56f803fc85d51f252abb1249
5595d2e815522ed4b3e3469a
565811ce16b6c7089cbc5a40


bottom user's word count graph
wordCount
readBy_predict
23.224006719317526
23.446453168044076
25.08148130787098
25.49023834282771
25.692497948032603
58bb512ad73408ce4f4e6de8
564a46ea16b6c7089cbaf217
591740e9d73408ce4f5f4d3f
5920504fd73408ce4f60f604
596359bdd73408ce4f6c6c5c



*due to the randomness of machine learning prediction, the graphs generated may be different every running the program

In the 12 graphs above, the x columns show the users ID of either top 5 user or bottom 5 user, and y columns show the tree language features (***mention count, exclamation ratio, polarity, word count, noun ratio, and subjectivity***)

From a brief overview of the set of comparison, we can discover that top APR users' messages are more likely to have smaller values for features like ***exclamation count*** and ***word count***, and have greater or positive values for features include ***polarity*** and ***subjectivity***. For ***mention count***, top APR users' messages are more likely to have it around 1 mention permessage and bottom APR users' message are more extreme regarding this feature. Most of time, their messages have no mention, but there are also some cases that mention count reaches extremely high. For ***noun ratio***, top APR users' messages are more evenly distributed between 5% to 10%, while bottom APR users' messages are widely ranging between 0% to 25%.

Thinking back to the title of this group chat, freecodecamp, we know that this group chat is a very common place for people to share their code with each other, therefore a good amount of messages will be pure coding language. After sum up all those features differences between top and bottom APR users' message, we think top users' message features are more resemble the features of human language while bottom users' message feature reminds us of coding language. It is common for code people shared to have greater word count than real text message, and sometimes even extremely long code (2500+ word count) might be shared. For the higher exclamation count, we tend to understand it as a common use of logical term in coding language. The lack of sentiment(polarity and subjectivity) also shows the features of coding language compared to natural language.

This feature analysis really shows an ironic result: even though, this group chat was named after 'code', but the code message people shared tend to be less popular compared to real human language messages. In conclusion, having a stronger personal opinion, and higher positivity will tend to make a message more popular. However, greater exclamation count and word count will have negative impact on message popularity. Last but not least, consistently mention one person and keep the noun ratio of message between 5% to 10% person also contribute to higher message popularity.

6. Reproducing the result

Obtaining the data and preparing the program

Download the zipped file from kaggle (requires account): https://www.kaggle.com/freecodecamp/all-posts-public-main-chatroom Unzipping it will reveal two csv files inside. We will be using the one named: *freecodecamp_casual_chatroom_anon.csv* in our project. Download the uploaded code zip from gradescope (or git clone from our repo). Take the csv mentioned earlier and put it in the data folder. You are now set to run the program

Running the program

Extra dependencies: textblob, graphviz

Running main.py will provide a command line interface which provides various options to interface. First type `freecodecamp` . Then you'll be given 6 options to choose from. Options 1-3 provide different percentages to run the dataset. Option 1-3 will also generate essential pickle files for analyzation and testing in later steps. Option 4 creates a model from a previously saved pickle. Option 5 runs the hyperparameters testing from a saved pickle. Option 6 runs the suit of feature analysis from a saved pickle. Type `1` to run 100% of the dataset (took 3-4 hours on my macbook w/ 16gb ram and 2.2ghz intel i7). Once the program completes, it will automatically output a pdf (opens automatically on my mac) of a graph of our model.

Also included is a groupme (serialized) dataset which may be a better representation of our programs potential. By typing `groupme` at the start and pressing enter to start, the program will construct a model of the groupme data.

Feature Analysis

To run the feature analysis graphs, you must type `freecodecamp` and then type 6. This run the entire feature analysis suite.

7. Work Plan Evaluation

In our work plan, we planned to have 5 days to work on cleaning data and generating features, 1 days to set and setup optimized machine learning, and 8 days to work on analyzing the result from machine learning.

Our cleaning data process went on smoothly, did not take as much time as we expect, mostly due to our good preparation and data exploration during project part 0 and part 1.

However, we significantly underestimated workloads in generating feature and machine learning testing. Due to the huge data size (about 5 million messages), trying to optimize the efficiency of our program so that running the whole dataset

will be less time and memory consuming was one of the biggest challenge we faced in this part. Beyond that, repeated machine learning test trying to improve our model's predicting accuracy is another huge time investment. Due to the low quality of groupBy column in our dataset to represent messages' popularity, it was significantly harder to make big improvement over our model in this project compared to the testing we have done in class before.

Beside, we also forgot to include time we need to restructure the whole project. In the last stage of our project, we spent most of our efforts trying to connect all our function through main file and make the whole project easily reactable to other users. A lot of debugging and function rewriting appeared in this part, which could have been avoided if we start with a more complete idea about our project's structure.

All these misestimation about time usage in the first two steps lead to an extreme time limit for our final analysis. We were only left with two days to work on that. As a result, the analysis might be one of the biggest setback from what we originally planned. Therefore, if we have any chance to keep develop our project in the future, a more through and in depth final analysis will be the main focus.

8. Testing

Testing cleaning and getting features

To run the series of tests, you have to type `freecodecamp` and type '`7`'. This will run a suite of tests to help verify whether the data was gathered properly. These tests check if the data cleaned contained the right columns whether the features produces the right columns and checks all of the gathered features to ensure they are within their correct ranges (non-negative word counts, etc). We used asserts to ensure the results were expected.

Logging and the Playground

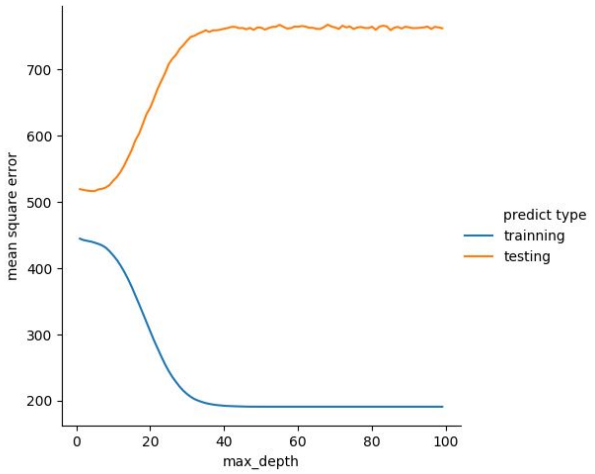
While we used our testing file to make sure the most sensitive parts of are code worked, we also heavily used the `logging` module and breakpoints to figure

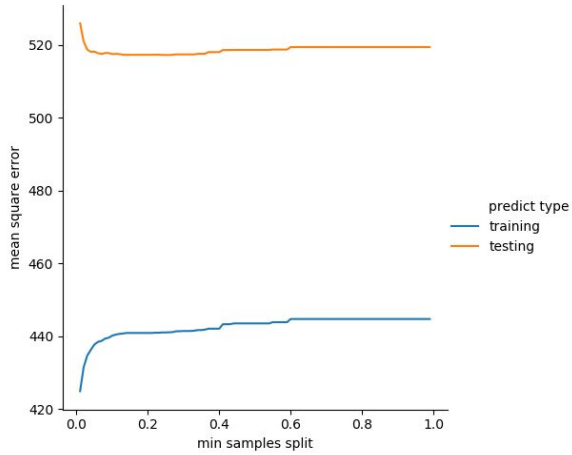
out where our code was going wrong. When our code runs, the log output is printed to the terminal indicated where the program was in processing the data and more. This was especially helpful when programs took more than 20 minutes to run and we needed to know where it was getting stuck. When figuring out what was going wrong when our program got stuck, we manually tested our code using breakpoints to play with our data in a interactive shell and test our data for specific points.

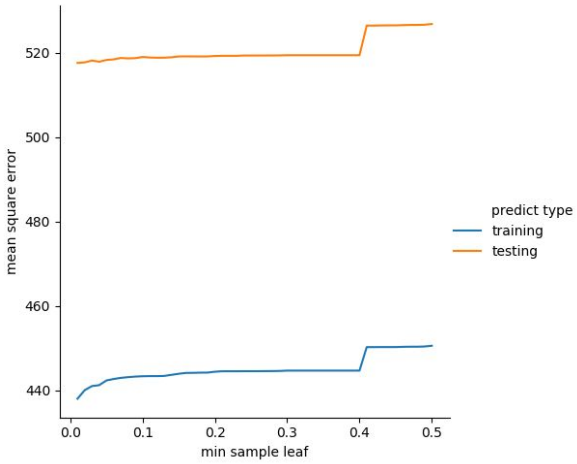
Machine learning Test

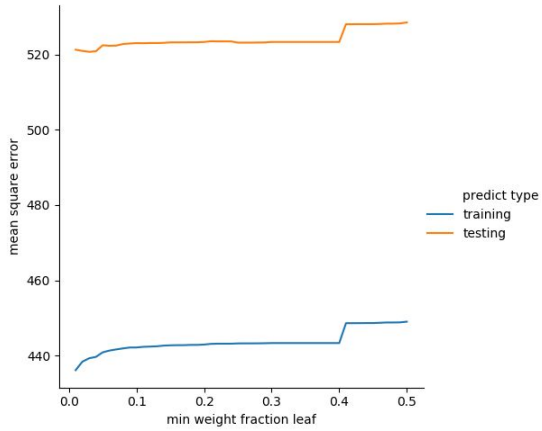
In this test, our goal is testing our machine learning model's various hyperparameters to find out the most important hyperparameters regarding model prediction's mean square error and what the optimized setting for those parameters so that we can minimized our model prediction's mean square error.

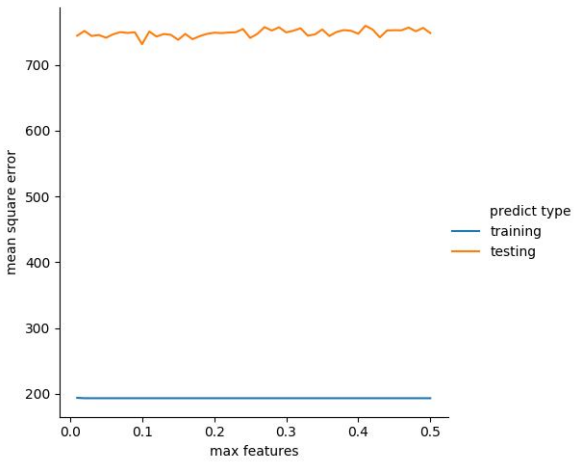
We created a file called machine_leaarning_test.py to perform this test. Inside of the file our first function is called graph_analysis which takes in 5% of the train set data and dev set data to train and test models built with different settings of decision tree regressor's hyperparameters, including: max_depth, min_sample_split, min_sample_leaf, min_weight_fraction_leaf, max_feature, max_leaf_nodes, and min_impurity_decrease. For every hyperparameters, it will print out the setting at minimum test mean square root and plot out a graph showing relationships between different setting and mean square error for each hyperparameters. Following are the graphs we generated:

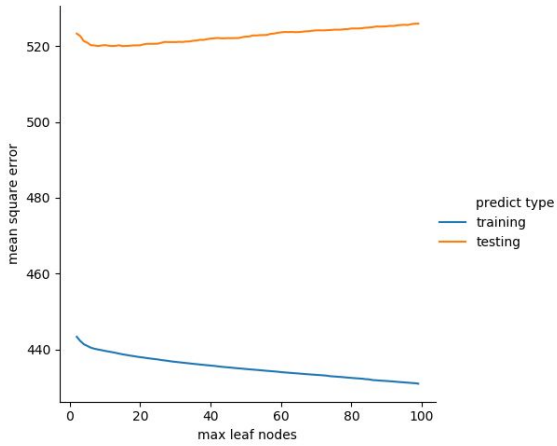

mean square error
700
600
500
400
300
200
0
20
40
60
80
100
max_depth
predict type
trainning
testing

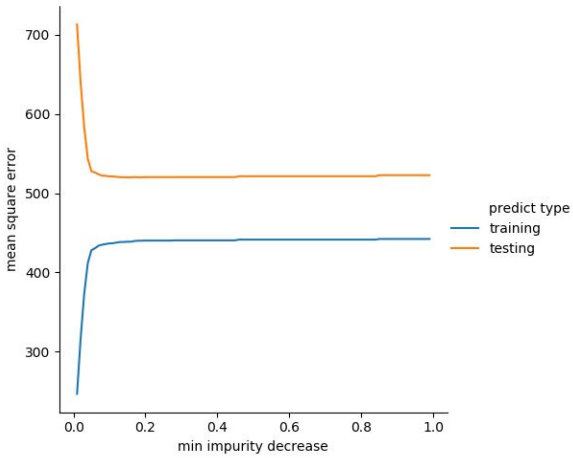

mean square error
520
500
480
460
440
420
0.0
0.2
0.4
0.6
0.8
1.0
min samples split
predict type
training
testing


mean square error
520
500
480
460
440
0.0
0.1
0.2
0.3
0.4
0.5
min sample leaf
predict type
training
testing


mean square error
520
500
480
460
440
0.0
0.1
0.2
0.3
0.4
0.5
min weight fraction leaf
predict type
training
testing


mean square error
700
600
500
400
300
200
0.0
0.1
0.2
0.3
0.4
0.5
max features
predict type
training
testing


mean square error
520
500
480
460
440
0
20
40
60
80
100
max leaf nodes
predict type
training
testing



After manually analyzing the graphs, we decided to pick max_depth, max_leaf_nodes, and min_impurity_decrease out of the eight hyperparameters to further continue the test.

In the second round of test, we wrote a method called isolated analysis which takes in all the train and dev datas. I it will loop over the range of each parameter with only ten steps and then choose the place with the lowest test mean square root to further looping a smaller range but smaller steps, and eventually it found out the optimum setting for all there hyperparameters.

In the last test, we will be testing the situation when each two pairs of hyperparameters are being applied together. By setting one parameter as the optimum setting and loop the other one in a range we find out does the optimum hyperparameter setting under an interactive situation.

In this end, we find out the optimum hyperparameter setting for our model is: max_depth=6, max_leaf_nodes=31, min_impurity_decrease=0.03.

9. Last Words

Our goal was to create a model that could predict the popularity of a message based on a given dataset. While our error was unbelievably high on the freecodecamp dataset (our primary dataset), our error was quite low on the groupme data set. We concluded that the `readBy` value from the freecodecamp dataset was not a good indicator of popularity but our private dataset's `likes` value

was. This does make sense with our hypothesis as `readBys` represented how many people read the post which may or may not be intentional. However, `likes` indicate an active push for popularity as the user must click the like button to like the message. This showcases that our model does indeed work and could work beyond this project.

Our project was developed using git and is available on github here:
https://github.com/RitikShah/cse163-project