Abstract

In recent times, Natural Language Processing as a field has grown tremendously in a variety of fields including but not limited to chat bots, autocorrect, autocomplete, and language translation. This project aims to build off of the midterm project, focusing on one movie series as opposed to two. There are three functionalities of the code base: character relationship visualization, character growth across the trilogy, and Machine Learning for text classification.

The *Lord of The Rings* dataset contained just 2 csv files with the core file being of size 229KB. From the *Lord of The Rings* dataset, only 'lotr_scripts.csv' was utilized in the analysis.

The final code submission contains two .ipynb files: one for the two types of visualizations, and the second solely dedicated to the ML tasks. The project was split into multiple phases: Data Cleaning, Data Visualization, and finally ML.

Table of Contents

I.Data Cleaning

In contrast to the midterm project where the focus was on data exploration, incremental data cleaning and rudimentary data visualization, the focus in the final project is for more complex data visualization and Machine Learning. Thus, it made more sense to streamline the process and convert the entire process to a set of functions. Thus we have the following set of functions used for the entire data cleaning process: *get_char_classes(), remove_voice(), fix_spelling(), clean_dialog(),* and *clean_df().*

The function *get_char_stats()* is a simple utility function that displays all the unique values in the 'char' column. It is a one line void function that takes as input the data frame object and the name of the column. Inside the function body, we call unique() on the series object and then len to get the number of unique character classes. In a print statement, we display the number of classes and the classes themselves. Before the data cleaning on the 'char' column we have 118 character classes. After the data cleaning, we have 85 character classes or roughly a 28% decrease.
Now we will move onto the functions that perform the actual data cleaning. The function *remove_voice()* manually adds the lines of characters that have lines but are not present on the screen and adds 'VOICE' as a character of its own called 'NARRATOR' to avoid confusion with a movie character.
Next, f*ix_spelling()* manually converts the various variations of characters to a standard character name. In order to perform some of the transformations, a prior knowledge of the series is required as in the case of 'STRIDER'. In hindsight, it would have been more efficient to use modules inside the nltk package to perform lemmatization. Since the movie script contains inconsistencies in spelling such as 'GANDALF' and 'GAN-DALF', it was important to consider the substrings in order to perform standardization.
We next move on to the *clean_dialog()* functions which utilizes the powerful nltk library and re library to prep the text for both the visualizations and machine learning. First, we define two regular expressions, REPLACE_BY_SPACE_RE and BAD_SYMBOLS_RE which together remove all the punctuation and non-word characters. From nltk, we

import the list of stopwords in the English language in order to filter them out from the text.

Finally, we conclude with the function *clean_df()* which applies the three previously described functions. This function takes as input the data frame and column to process. The only columns that are processed are ‘char’, ‘movie’, and ‘dialog’. First we use *upper()* to uppercase every character. Next we replace any punctuation with the empty string. Next, we first remove garbage characters using the built in *strip()* function. Next, we apply the previously discussed *remove_voice()* function. Finally we apply the *fix_spelling()* function(). In the movie processing section, only upper casing and trimming using the *strip()* function are necessary. Finally, in the dialog processing section, the previously discussed *clean_dialog()* function is applied. It is observed that as a result of the application many character lines are shortened to either empty strings or extremely short lines. In order to bypass that issue, the data frame is filtered to contain only entries with at least 6 characters.

After preprocessing, we use the pandas *value_counts()* function to extract the top 10 characters. In order of the number of lines, the top 10 characters are: Frodo, Sam, Gandalf, Aragorn, Pippin, Merry, Gollum, Gimli, Theoden, and Faramir.

II. Data Visualization

In a shift from the midterm project, the final project involves breaking down the data frame by movie. As such, the first three steps in the visualization section are to extract three data frames, compute the top ten characters from each movie, and display the top top characters by line with their counts for each movie. **Figure 1** shows the breakdown for each of the movies. Based on the figure, after preprocessing, the first movie is left with only 8 characters with speaking lines compared to the last two movies. In the second movie, Gandalf has the fewest number of lines in the second movie because he dies fighting the Balrog in Moria at the end of the first movie. He gets resurrected by Eru Iluvatar(God) and comes back some time later in the second movie. By the last movie,

Gandalf returns to the top of the rankings with the most number of lines. Since Gollum is only introduced at the end of the first movie, he has the fewest lines. However, as the fellowship gets broken up, Gollum becomes Sam and Frodo's guide. In the last two movies, Faramir appears as the character with the fewest lines in the last two movies.

char	obs
GANDALF	65
FRODO	58
ARAGORN	45
MERRY	32
SAM	32
PIPPIN	24
GIMLI	18
GOLLUM	3

Movie #1: 8 chars

char	obs
ARAGORN	85
SAM	81
GOLLUM	68
FRODO	66
THEODEN	59
GIMLI	51
GANDALF	46
MERRY	44
PIPPIN	42
FARAMIR	34

Movie #2: 10 chars

char	obs
GANDALF	84
SAM	77
FRODO	58
PIPPIN	52
ARAGORN	49
GOLLUM	47
THEODEN	39
MERRY	33
GIMLI	30
FARAMIR	22

Movie #3: 10 chars

Figure 1. Breakdown of top 10 characters by the number of lines

For the last two visualizations, the following three functions were written: *consolidate_char_vocab(), get_relationships(),* and *get_char_growth().* Most of the code in *consolidate_char_vocab()* was recycled from the midterm project. However, given that Faramir appears only in the second movie and Theoden only appears in the third movie, it was necessary to create a condition in the function to check if the character vocabulary extracted from the 'corpus_dialog' column(created by applying the Counter() function) was empty. If that was the case, a dummy dictionary containing 5 dummy words each with a count of zero was to be returned. That way, when generating the histogram in the function *get_char_growth(),* the introduction of the two characters could

be depicted visually. If the character vocab was non-empty, the mega dictionary was generated by consolidating the dictionaries created for each line of character dialog.

Now that the method obtaining the character corpus has been completed, we can move on to discussing the *get_relationships()* function. This function takes in the movie dataframe, list of top 10 characters, and the threshold that defines the minimum number of times a character has to reference another character for us to deem that a character relationship. The networkx Python library is used to generate the network. **Figure 2** depicts sample graphs generated as a result of a call to *get_relationships().* The figure contains two networks depicting both acyclic and cyclic networks. Perhaps if I had more time, I would examine why the cycles are depicted. In general, it does not make sense why any character other than Gollum would talk about themselves in third person. Based on the figure, there exist characters that refer to themselves in the third person up to five times. **Figure 3**

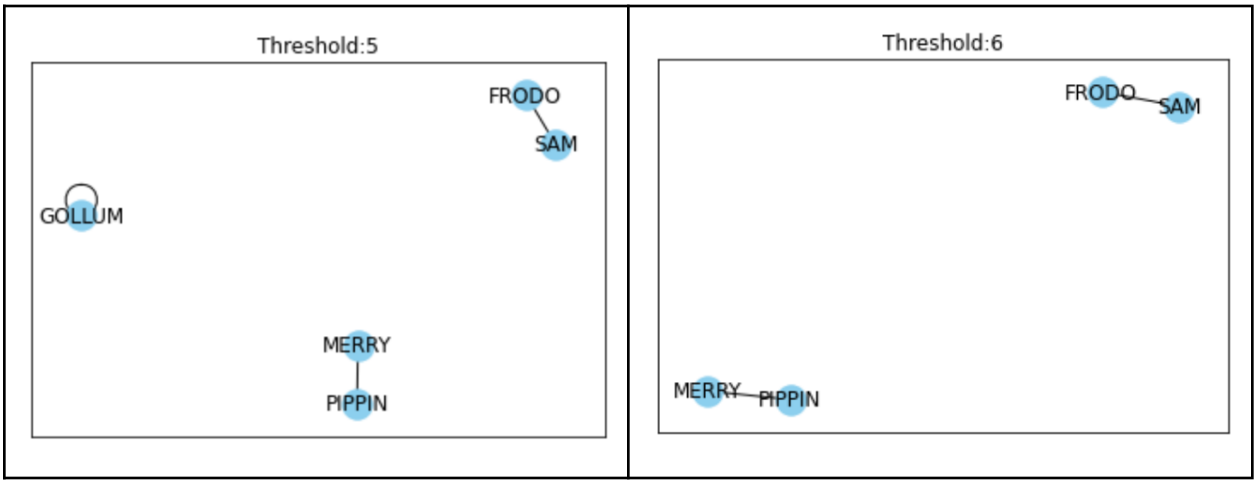

Figure 2. Relationship graphs for top 10 characters in the movie *The Two Towers.*

Now let us move on to the function *get_char_growth()* which takes in the list of the three dataframes, the character for which we are visualizing the growth, and the index that helps choose a unique color for each character from a list of 10 colors. For each movie in the dataframe list, the character vocabulary corpus is extracted. From that, the top 5 words are extracted. The corpus is then filtered to only include the key-value pairs associated with the top 5 words. The keys and values are extracted separately and used to create the bar plots. **Figure 3** showcases the character growth for a character that has appeared in the entire trilogy like Frodo and one that was introduced later like Theoden. Based on the figure, Frodo's vocabulary shows that he consistently relies on Sam throughout the trilogy. In the first movie, he also interacts with Bilbo and Gandalf. In the second movie, he is forced to spend a lot of time with Smeagol(Gollum) because Sam and Frodo are split from the fellowship and rely on Smeagol to lead them to Mount Doom in Mordor to destroy the ring.

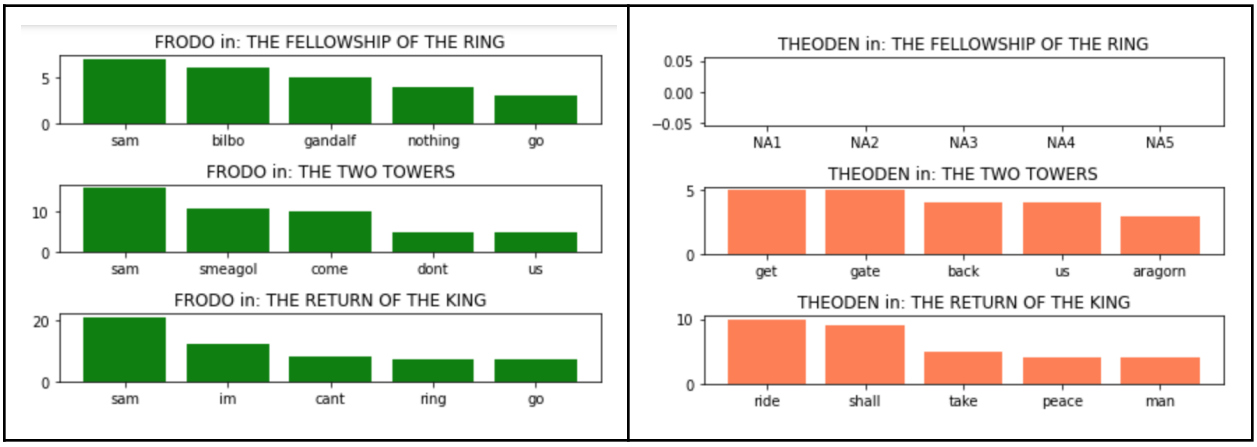

Figure 3. Character growth for Frodo and Theoden

III. Machine Learning

The second ipynb file contains the entire machine learning section. We will discuss the three functions: *best_model_stats(), get_best_params(),* and *get_cv_plot().* The function *best_model_stats()* is primarily used to depict the confusion matrix as the sample in **Figure 4** shows. The function *get_best_params()* uses the RandomizedSearchCV module from scikit learn model selection to pick the model parameters that maximize the accuracy metric. The parameters tuned include the number of estimators, max_features, max_depth, min_samples_split, min_samples_leaf, oob_score, and class_weight. The cross validation is set to 5. Finally, the function *get_cv_plot()* uses the *cross_val_score* to perform cross validation on the training data and generates a line plot for accuracy scores.

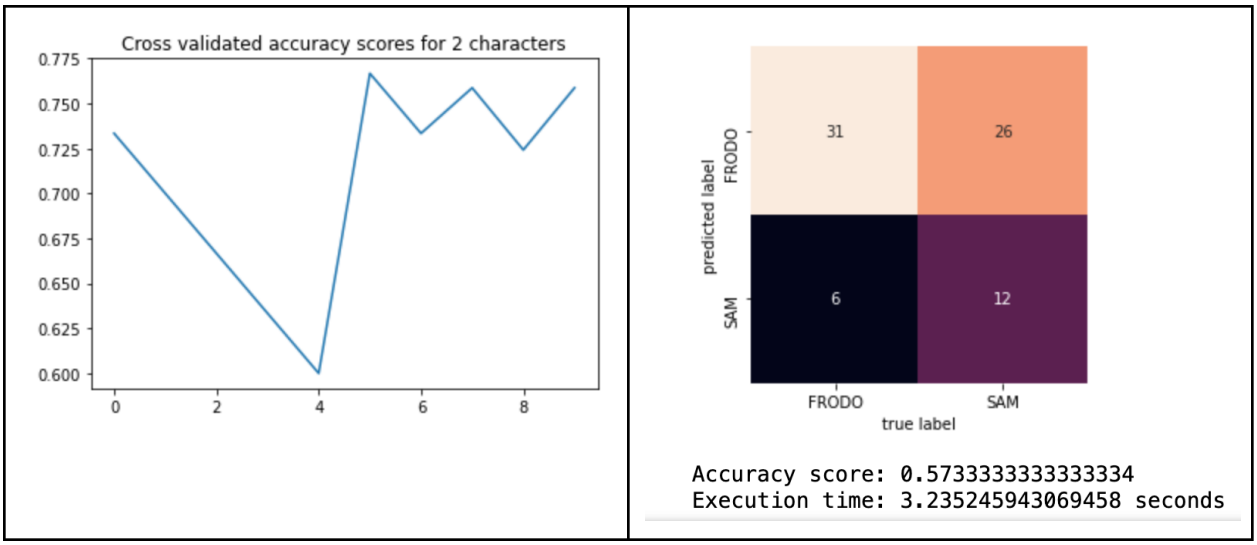

Figure 4. Output of *get_cv_plot()* and *best_model_stats()* for two character classes including Frodo and Sam

Although not officially a function, the last bit of logic utilizes the three functions previously discussed. First, we perform preprocessing on the data frame. This involves dropping any NaN values that may be introduced, filtering the data frame to include only

entries associated with a specific subset of top 10 characters, transforming the dialog using the TfidfVectorizer(), one hot encoding the character classes and splitting the data frame into train and test sets. The reason TfidfVectorizer was chosen was to deal properly with rare words and common words by assigning penalties.

Once the train and test sets are created, the *get_best_params()* function is called to extract a dictionary of the best parameters. The dictionary entries are next fed to a RandomForestClassifier. Next *get_cv_plot()* and *best_model_stats()* are called to visualize the results of ML. It is important to note that the cv plot displays primarily the problem with this task. Based on the current methodology, the vectorized dialog is not statistically significant enough of a feature to be able to predict which character said a certain line. This is proven by the erratic nature of the accuracy score for every model trained. Depending on what lines are included, the classification accuracy increases or decreases. In **Figure 4,** The best cross validation accuracy score is over 75% but the accuracy outputted by *best_model_stats()* is at 57.3%, a difference of almost 18%.

IV. Conclusion

The focus in the final project on meaningful visualizations and Machine Learning proved to be an incredible learning experience. Oftentimes, we try to throw Machine Learning as a solution to many problems but it is not always the best solution. In this case, the visualizations provided more meaningful insights into the data.

Sources:

1. https://www.kaggle.com/paultimothymooney/lord-of-the-rings-data