Developing Predictive Language Models for *The Office* Script

LING 227 Final Project
Spring 2021

Luke Benson
luke.benson@yale.edu

Karen Jiang
karen.jiang@yale.edu

Abstract

We propose several language models for the script of *The Office* that aim to predict the character speaking a given line of dialogue within the script. We engineered various text features that we then utilized to develop four different multinomial logistic regression classifiers. Our final model was able to achieve 44.1% accuracy on the training set and 34.7% accuracy on the test set. From the feature weights in our models, we were also able to detect various character relationships and plotlines that are present in the show's narrative arc.

1 Introduction

The Office is an American television series that aired from 2005 to 2013, cementing its place in TV history as one of the most popular shows in the 21st century. Throughout the series, the characters of the *The Office* have distinct speech patterns and follow unique plot lines, providing an impetus to build predictive language models for the script. The goal of this study is to develop logistic regression models that can predict the character speaking a given line of dialogue using various text features.

Previous researchers have conducted exploratory data analyses on *The Office* but have not yet attempted to build a predictive model. Jenna Allen discovered varying correlations in word frequencies between different characters (Allen, 2018), Kristóf Rábay conducted sentiment analysis on the script and found significant differences in average sentiment among characters (Rábay, 2020), and Swarnita Venkatraman analyzed character relationships throughout the show's nine season (Venkatraman, 2020). Although these previous researchers have not built predictive models, their analyses suggest that there are significant differences in character speech patterns, further motivating our study.

2 Approach

2.1 Data

The script of the *The Office* has been documented and processed by previous researchers and is available through the `schrute` package (Lindblad, 2021). The dataset covers all 55,130 lines from all nine seasons of *The Office*. For each line, the dataset contains its season, episode, episode writer, episode director, character speaking, dialogue as text, and dialogue with direction (i.e. [on the phone] or [looks at Pam]).

Over its nine-season history, *The Office* introduces more than 700 characters. While we aim to use as many of the lines in the full script as we can, it would be impossible to train a machine learning algorithm to correctly predict, for example, the single line for "Guy buying doll" or "Male applicant 2." In order to make accurate predictions, our models must be able to develop some kind of understanding of a character's vocabulary and speech patterns. Therefore, we restricted our classification task to the show's twelve primary characters: Michael, Dwight, Jim, Pam, Andy, Angela, Kevin, Erin, Oscar, Ryan, Darryl, and Phyllis. These characters speak 41,906 of the 55,130 total lines. The distribution of lines across these these twelve characters is shown in Figure 1.

To make the lines more useful for classification, we performed multiple text cleaning operations on our script. Given that numbers of a certain value appear maybe once or twice throughout the entire script, we replaced each number with just "numeric." To reduce the number of unique words in the script, we also stemmed every word using the `PortStemmer` package in `nltk`. Stemming takes each word to its root: "abandoned," "abandons," and "abandon," for example, all become "abandon."

Of the remaining 41,906 lines, we used 80% as

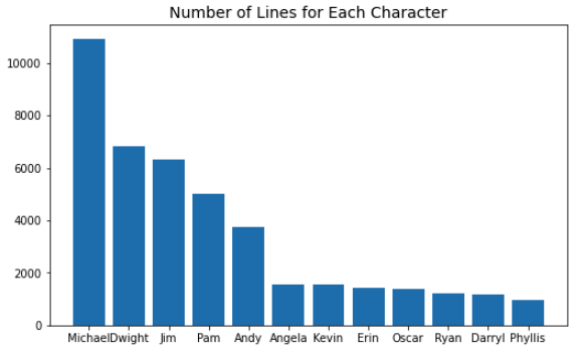

Figure 1: Distribution of lines among the top twelve characters.

training data (n_{train} = 33,906) and reserve 20% (n_{test} = 8,000) for testing. Splitting our data into a train and test set allows us to gauge how well our models generalize to data that they have never seen before.

2.2 Algorithms

We developed several multinomial logistic regression models for this classification task. Whereas standard logistic regression models are trained to predict one of two possible classes, multinomial logistic regression generalizes to cases in which we have $k > 2$ possible outputs. For each observation, the model outputs a probability distribution over these k classes, and the predicted class for that observation is that which corresponds to the highest probability. Here, we had $k = 12$ possible characters.

In addition to being widely-used within the technical industry, logistic regression also lends itself to interpretable results. Similarly to linear regression, a logistic regression model learns weights for each of the parameters in the input feature matrix. In the case of multinomial logistic regression, the model learns a set of feature weights for each of the possible k outputs. If each of the features have been standardized to the same scale, then we can assess the relative importance of features by simply comparing their learned feature weights: features which are more significant to the outcome of our model will have larger weights.

3 Feature Engineering

3.1 Non-Text Features

Our goal was to use the text within each line to predict the speaker of the line. However, we also considered incorporating a limited number of non-text features into our models. These non-text features could provide important context to the script which the line text alone cannot.

Season: The season in which the line occurred. For a character like Erin, who doesn't appear until season 5, this information would be valuable.

Previous Speaker: The speaker of the previous line in the script. This feature could be useful since some characters, such as the group of office accountants (Angela, Kevin, and Oscar), are more often in conversation with each other. If the line is the first in an episode, this was labeled as 'Start.' Given the randomness of characters outside the main twelve characters we focus on, we ignore the previous speaker feature if the previous speaker was 'Other.'

3.2 Text Features

Outside of the words themselves, we suspected that there might be certain features within the lines of the script which could give us insight into the character that spoke them. For example, some characters may be more likely to have lines which are longer, ask more questions, or say more non-grammatical "gibberish" words.

Line Length: The number of words in the line.

Numeric: A number (any number) is present in the line.

Question: A question mark is present in the line.

Exclamation: An exclamation point is present in the line.

Director: A director's note is present in the line.

Ellipses: Ellipses are present in the line.

Dash: A dash (—) is present in the line.

Gibberish: A "gibberish" word is present in the line. "Gibberish" words are those which contain three or more of the same character in a row, e.g. "aaaagh" or "aaaaeeeexcellent." The proportion of each character's lines that contain gibberish are shown in Figure 2A.

Sentiment: The sentiment of the line, as calculated by the `SentimentIntensityAnalyzer` package in `nltk`. The average line sentiment for a given character's lines in a given season is shown in Figure 2B.

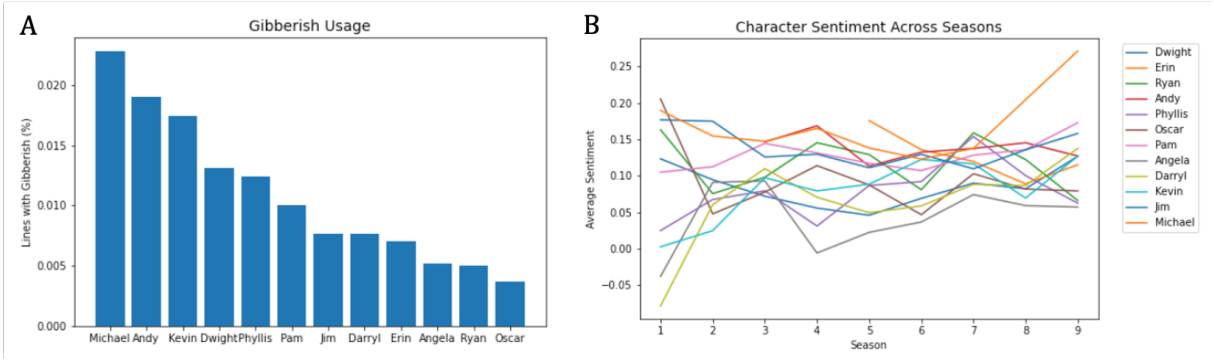

Figure 2: **(A)** Gibberish usage among the top twelve characters, reported as a percent of their total lines spoken. **(B)** Average line sentiment in a given season among the top twelve characters.

3.3 Unigram Features

Up until this point, we had yet to consider the actual words in each line. We could have included an indicator feature for each of the $15,885$ tokenized words in our script, each data point being a 1 if the word is present in a line and 0 if not. However, since most words are absent from any given line, this approach would result in an incredibly sparse and ultimately uninformative feature matrix.

Instead, we considered using the script words, or unigrams, to calculate the probability of each line occurring given a particular character: $P(line|character)$. We called these probabilities our line probabilities for each character.

Calculating line probabilities for each character was done in 3 steps: (1) constructing a matrix of character unigram counts from the training set, (2) calculating $P(unigram|character)$ given these character unigram counts, and (3) calculating $\log(P(line|character))$ given $P(unigram|character)$ and the presence of unigrams in each line.

We can illustrate this process for a given character, say Jim. In step (1), we added up the total number of occurrences of each unigram throughout all of Jim's lines. For step (2), we could have calculated the probability of a specific unigram, $P(unigram|Jim)$ by simply adding up the total number of occurrences of that unigram and dividing it by the total number of words that Jim speaks throughout the show. However, with this approach, some of the resulting probabilities would equal 0. Even though, for example, Jim never says "affection" in our training set, there would still be some small chance that he says it somewhere in the test set. Therefore, we wanted to avoid setting $P(\text{"}affection\text{"}|Jim) = 0$.

To adjust for this consideration, we utilized Laplace smoothing: we added 1 to every unigram count for Jim, and then calculated $P(unigram|Jim)$ by dividing those counts by Jim's total number of words plus our vocabulary size, V. The final probability for our given unigram, "affection," appears in the following form:

$$P(\text{"}affection\text{"}|Jim) =$$

$$\frac{count(\text{"}affection\text{"}, Jim) + 1}{\sum_{unigram} count(unigram, Jim) + V}$$

Now that we had $P(unigram|Jim)$ for every unigram based on our training data, we could calculate $P(line|Jim)$ for step (3) simply by multiplying $P(unigram|Jim)$ for each unigram in a given line. However, to make our computation easier, we added these probabilities rather than multiplying them by taking the log of each $P(unigram|Jim)$. Upon summing $\log(P(unigram|Jim))$ for each unigram in a given line, we arrived at our desired probability, $\log(P(line|Jim))$. We completed this step for all lines in the train and test set.

We then carried out this process for all twelve characters, and thus ended up with a set of twelve variables, $P(line|character)$ for each character. We were now able to incorporate these variables into our logistic regression feature matrix.

4 Modeling

4.1 Model Design

When running our models, we employed the `LogisticRegression` package from `sklearn`. This package allows for the selection of multiple parameters in logistic regression, including the optimizer type, the regularization coefficient, and the maximum number of gradient

updates. While our goal was to predict the speaker of each line with as high a degree of accuracy as possible, we were ultimately also interested in how models performed relative to each other. Thus, we kept the all of these parameters the same across our different models.

We used $sklearn$'s "newton-cg" solver, which implements Newton's Method for optimization. Newton's method is a version of gradient descent in which we also include a matrix of second-order partial derivatives (called the *Hessian*). While it is computationally expensive, this gradient update method generally works very well for multinomial logistic regression problems.

Regularization essentially caps the magnitude of the learned feature coefficients, and generally leads to better performance on test data. However, without a validation set to tune our hyperparameters, we chose to leave our models unregularized instead of choosing an arbitrary regularization coefficient. Finally, to allow our models to train fully until convergence, we set the maximum number of gradient updates to be 1,000 instead of the default = 100.

We tested four different models in predicting the character who spoke each line. These four models varied only in the set of features used.

Model 1: *Baseline model*. Non-text features only (season and previous speaker).

Model 2: Unigram probability features only.

Model 3: Non-text features and text features.

Model 4: All features previously described: non-text, text, and unigram probabilities.

4.2 Results

The performance of our models on both the training and test sets is reported Table 1. We measured performance using two parameters: accuracy and weighted F1 score. We chose the weighted F1 score because our data is not evenly distributed across characters.

We also visualized the results of our models using confusion matrices that display the percentage of a given character's lines that are predicted correctly by the model. A model with perfect accuracy would show only yellow squares along the diagonals. These confusion matrices are shown in Figure 3.

Looking at the results in Table 1 and confusion matrices in Figure 3, we see that our final model yielded the best accuracy and weighted F1 score and also performed best in terms of observing the ideal diagonal along the confusion matrix (Figure 3D).

Model	Data	Accuracy	Weighted F1 Score
Model 1	*Train*	0.2928	0.2314
	Test	0.2934	0.2327
Model 2	*Train*	0.4151	0.3517
	Test	0.3100	0.2388
Model 3	*Train*	0.2995	0.2407
	Test	0.3030	0.2434
Model 4	*Train*	0.4414*	0.3932*
	Test	0.3468*	0.2968*

Table 1: Performance of our four models on the training set and test set: Model 1 (non-text features), Model 2 (unigram probability features), Model 3 (non-text and text features), Model 4 (all features). ***** *Indicates best performance among the models*

Our baseline model, whose only features were season and previous speaker, was able to achieve around 29% accuracy on both the training and test sets. Given that Michael speaks 26.0% of the total 41,906 lines among our twelve characters, it appears that adding season and previous speaker is able to achieve an additional 3% accuracy over guessing "Michael" as a default.

The unigram model (model 2), achieved 41.5% accuracy on the training set and 31.0% accuracy on the test set, which is a significant improvement on training accuracy but not on test accuracy. The confusion matrix for model 2 (Figure 3B), begins to show evidence of a diagonal but also shows evidence of over-predicting "Michael," as suggested by the density of Michael's prediction column. The phenomenon of over-predicting "Michael" is likely due to the fact that he has the most lines and therefore the largest vocabulary, which subsequently means that he has the highest log probability for most words and thus most sentences.

Our engineered text features (model 3) do not produce a significant improvement in either training or test accuracy over the baseline model. However, when these features are combined with the unigram probability features (model 2) to create the full model (model 4), we do observe a 3% increase in both training and test accuracy beyond model 2. This suggests that our engineered text features provide additional context that is helpful when used in tandem with the unigram probabilities. The full model shows the strongest diagonal in its confusion matrix (Figure 4D), with improvements across all characters.

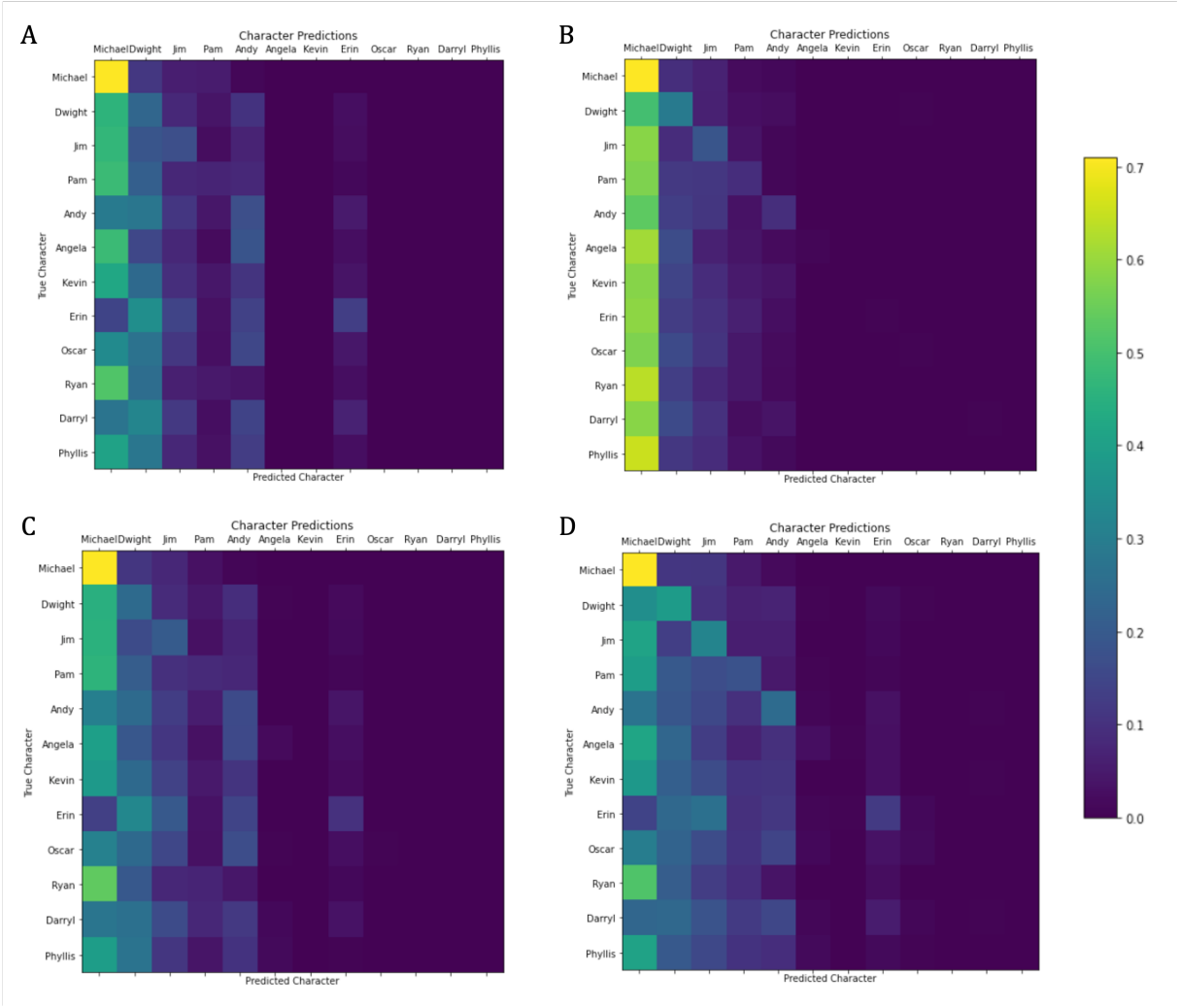

Figure 3: Confusion matrices of our four models' test set predictions. **(A)** Model 1 (non-text features), **(B)** Model 2 (unigram probability features), **(C)** Model 3 (non-text and text features), **(D)** Model 4 (all features).

5 Discussion

5.1 Weight Interpretation

We further investigate the weights of our various features in each model to better understand their specific contributions. The feature weights for specific character models are shown in Figure 4 for illustrative purposes.

From these plots we glean several insights:

1. The unigram log probability for a given character is by far the most important feature for that specific character. In the full model, the unigram probability eclipses the weight of our engineered features and non-text features (Figure 4D and Figure 4F).

2. Line length and season are the most important non-unigram features in the final model (Figure 4A). As seen in Figure 4B, season is especially important for characters who are not in all nine seasons. These include Erin (who is introduced in season 5) and Michael (who leaves the show after season 7, except for a single cameo in the show's finale in season 9). Erin also has the highest accuracy among the supporting characters outside of the top five characters, as seen in the full model's confusion matrix (Figure 4D), likely because she only appears in episodes from season 5 onward.

3. Character relationships can be seen through some of the features weights. For Pam (Figure 4E-F), the previous speaker being Jim has a high weight, given that Jim and Pam interact frequently throughout the show and have one of the most deeply explored character relationships. Likewise, for Kevin (Figure 4 C-D), the previous speaker being Angela or Oscar both have high weights, which makes sense because those three characters

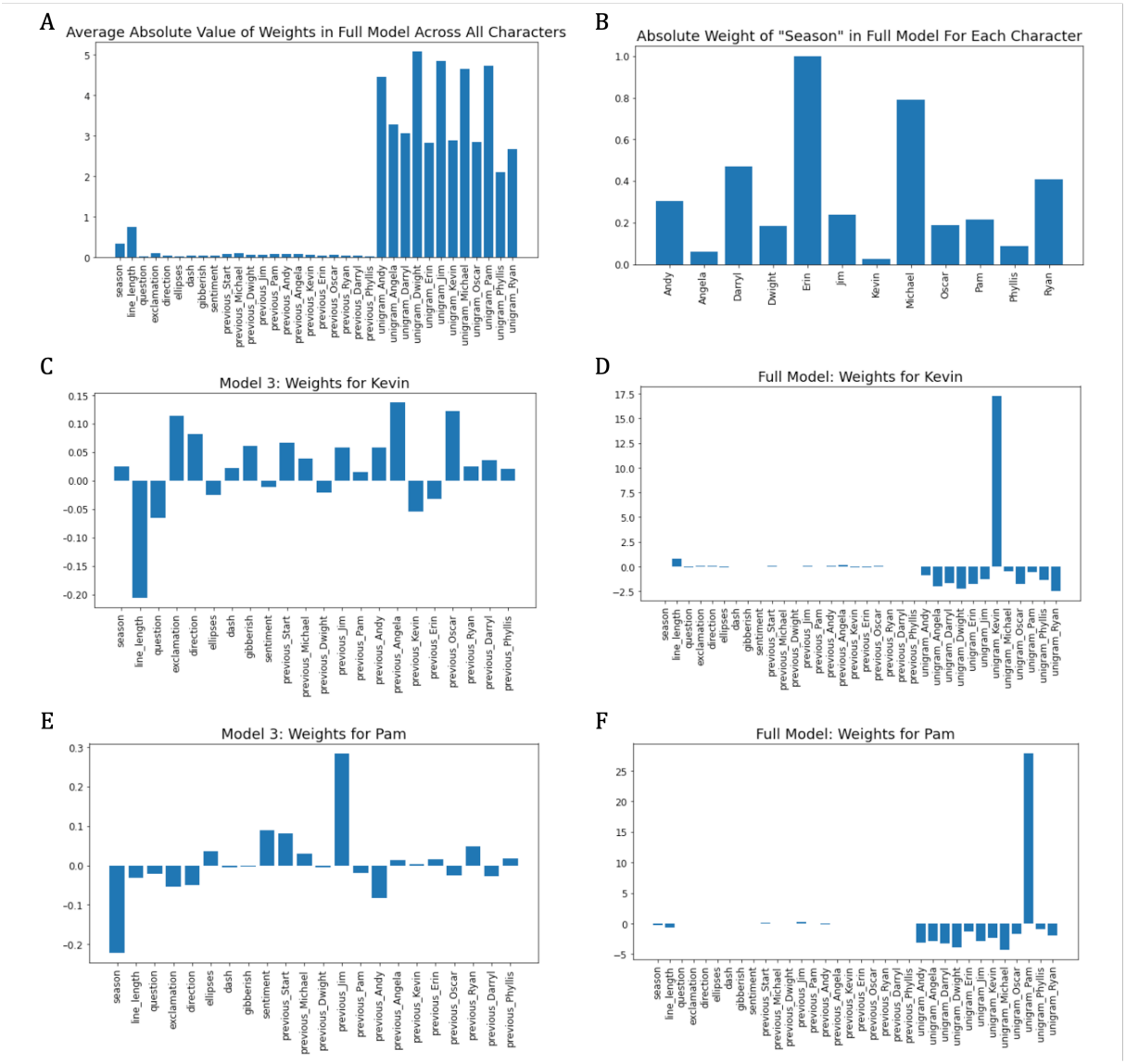

Figure 4: Feature weights. **(A)** Average absolute value of weights in full model, **(B)** Absolute weight of "Season" in Full Model" **(C)** Weights for Kevin (Model 3), **(D)** Weights for Kevin (Full Model), **(E)** Weights for Pam (Model 3), **(F)** Weights for Pam (Full Model).

work in the same division of the company and share many scenes.

5.2 Alternative Methods

The most obvious opportunity for future research would be to go beyond unigrams and incorporate bigram probabilities into our logistic regression model. This would allow our model to account for the fact that characters not only have different vocabularies, but also speak different two-word sequences, such as "love you" or "she said" (particularly important for *The Office*), at different rates.

We actually attempted to include bigrams probabilities as features in our model, but were unable to store the information for long enough to work with. With roughly 40,000 lines and 150,000 bigrams in the data set, we did not have the computing power to hold on to 6 billion data points, even using sparse matrices. Perhaps in the future, we could include a limited amount of bigram probability information, storing only the bigrams which appear most often in the script.

Additionally, when running our models, we might want to consider using a validation set and a test set. Having three distinct sets would allow us to train a model on one set, tune hyperparameters such as the optimizer and regularization coefficient on another set, and then test the best model on the final set. This process would ideally lead to higher accuracy in our models. However, for the sake of

simplicity in this project, we stuck to just a train and test set.

6 Conclusion

Overall, we present a multinomial logistic regression model that is able to predict the speaker of a given line of dialogue with significantly higher accuracy than randomly guessing the character with the most lines. This predictive model expands upon existing research, which, to-date, only includes exploratory analyses of the script. From the feature weights within our final logistic regression classifier, our model is also able to detect character relationships and plotlines central to the narrative of *The Office*.

References

Jenna Allen. 2018. Text mining: Every line from the office.

Brad Lindblad. 2021. *schrute: The Entire Transcript from the Office in Tidy Format*.

Kristóf Rábay. 2020. Nlp on the office series.

Swarnita Venkatraman. 2020. The office story: That's what the data said.