The Office: Text Mining and Natural Language Processing

Leticia Han

Math 533 Fall 2020

Abstract

The tremendous amount of text data made possible by continuous advancements in technology led to an explosion of methods and techniques allowing for sophisticated and insightful data analysis by combining natural language processing and machine learning. In this paper, we use a transcript of a television show called The Office to demonstrate how statistics and machine learning can be applied to textual data, including sentiment analysis, clustering, and regression.

1 INTRODUCTION

Since the invention of the Internet in the 1960s and the subsequent creation of the World Wide Web in 1989, technology has evolved dramatically allowing for an unprecedented amount of text data to be generated and collected for in-depth analysis in the form of text mining and natural language processing (NLP). Text mining is the process of extracting information and patterns from unstructured data, and it encompasses methods in statistics, computer science, and computational linguistics [5]. NLP is a way to enable computers to process and understand human language, verbal and written, and its techniques are heavily utilized in text mining [16]. It is an attempt to represent text with a richer semantic meaning and has been vital in deep learning.

Text mining tasks involve a large collection of methods, including classification, regression, clustering, generative language models, information extraction or retrieval, and text summarization [4, 5, 17, 24]. These tools are employed on text data that come from various sources, such as customer reviews, news, social media, novels, scientific journals, emails, and webpages. In recent years, an enormous amount of research and applications have emerged from a wide range of sectors, including science, marketing, entertainment, financial, legal, and humanities, demonstrating the significant value and increasing interest in analyzing textual data.

One example is an open Kaggle competition, CORD-19, with the task of text mining a large collection of research papers regarding COVID-19 to "generate new insights in support of the ongoing fight against" the disease [10]. Gentzkow et. al provide a survey of applications using text as data in economics and social sciences fields [17]. As an example, they describe a study done on the appendix of a book written by statistician Philip Wright to infer authorship attribution of that particular section. The speculation that

his son Sewall Wright was the true author of the appendix arose due to the fact that economists sensed a stark contrast between the appendix and the rest of the book. Using principal component analysis and clustering, the researchers found overwhelming evidence that the disputed appendix was most likely written by Philip and not Sewall. In another paper, Francis applies clustering to simulated text data from general liability claims file with injury description to demonstrate the use of text mining in actuarial and insurance sector [14]. With meaningful number of clusters found, he predicts the likelihood of a claim being a serious claim using logistic regression with attorney involvement and the injury variable derived from the clustering task as predictors. In addition, Allahyari et al. provides great introductory material to text mining and its application in the biomedical and health care domains, including clustering and classification [5]. Other applications include bestseller novels, social network analysis, sentiment analysis, stock price prediction, handwriting recognition, machine translation, and part of speech tagging among many others [9, 12, 13, 28, 40, 41, 55].

In this paper, we explore sentiment analysis, clustering, and regression using the transcript of a popular television series called *The Office*. The objective is to demonstrate applications of these methods to text data to draw insights from the show and predict ratings for the episodes.

1.1 Literature Review

Text mining is evidently an important and very active field of research with a tremendous body of literature that is growing continuously. There is no shortage of text data, research, and applications as shown by the diverse array of applications mentioned above. Text mining has gained huge traction in the film industry in recent years. Here, film is defined as both movies and television shows. For a multi-billion dollar industry, it is vital to understand how films are received by the audience in order to allocate resources efficiently and make decisions that can potentially affect writers, directors, actors, studios, investors, and all who are involved in the process from pre-production to final release. It has been shown text mining can offer valuable insights using machine learning, statistics, and natural language processing.

Although a large proportion of the current literature involves analyzing film reviews as text to predict viewer sentiment, movie revenues, or success of a film [3, 6, 15, 22, 26], there has also been some prior work on mining film scripts for analysis and prediction. Murtagh, Ganz, and McKie analyzed the transcript of the film Casablanca and scripts from six episodes of CSI (Crime Scene Investigation) using correspondence analysis and hierarchical clustering to validate the importance of narrative in screenwriting [37]. They noticed that while screenwriting for cinema movies have been the subject of many studies, there is a lack of literature on scripts for television. They argued that the unique properties of television serial dramatization necessitate tools to quantitatively support script writing using statistical models.

Another study proposed a story-based taxonomy of the movies by using k-means clustering to measure similarity among movie narratives based on movie character networks and genres of the movies mined from movie scripts [31]. Text mining movie synopsis to predict genre and rating of a movie has also been proposed for foreign movies, where the authors compare the performance of convolutional and recurrent neural networks with various embeddings for inputs to that of traditional algorithms, such as SVM and Random Forests, and showed that the deep learning models outperformed all the other models [6]. Recently, Vecchio et al. published a paper that proposed an innovative way to predict box office success using sentiment analysis of movie scripts to obtain the emotional arcs of the narratives and, thus, clustering the arcs to create a variable for their model [52]. Nemzer and Neymotin used linear regressions and recurrent neural networks to examine the relationship between movie descriptions and movie success

and found some genres negatively influenced movie success [39]. In addition, Netflix has been rumored to mine movie scripts for attributes and story arcs to predict viewership on the platform [29].

Although much of the literature on text mining of film scripts are focused on movies, methods and tools can be extended to TV scripts. An interesting paper was published in 2016 that proposed Text Recap Extraction Model (TREM) for TV series building on generic text summarization algorithms. It is an unsupervised model that extracts texts from plot descriptions to both summarize the current episode and prompt story development of the next episode [54]. Hunter et al. published a paper on predicting ratings for new television series by analyzing pilot episode scripts [23].

1.2 Dataset

The dataset used throughout this paper is from the `schrute` package available in R, containing the complete transcript for the NBC television show called *The Office* [33]. This popular sitcom which first aired in 2005 is an American adaptation of the British TV show by the same name, starring Steve Carell, Rainn Wilson, John Krasinski, Jenna Fisher, and B.J. Novak among many others [1]. Filmed in a distinctive mockumentary style, it follows the lives of the employees at the Scranton, PA branch of a fictional paper company called Dunder Mifflin. The show last aired in 2013, culminating with a total of 9 seasons and 201 episodes. In the dataset, two-part episodes were combined into one, resulting in only 186 episodes.

The dataset is composed of 55,130 rows, representing each line spoken by a character, and 12 variables. In this paper, we focus only on 8 variables by excluding identifier variables — index, episode name, air date — and text with stage directions (Table 1). The number of episodes and average IMDB rating for each season is outlined in Table 2. Season 1 had the smallest number of episodes with season 5 having the largest number of episodes. In addition, seasons 3, 4, and 5 were given the highest average ratings with seasons 1, 8, and 9 receiving the lowest average ratings throughout the show. The main character, Michael Scott played by Steve Carell, leaves the show in season 7, which might explain the decrease in ratings in the last two seasons (Fig. 1).

index	season	episode	episode_name	director	writer	character	text	text_w_direction	imdb_rating	total_votes	air_date
1	1	1	Pilot	Ken Kwapis	Ricky Gervais;Stephen Merchant;Greg Daniels	Michael	All right Jim. Your quarterlies look very good. How are things at the library?	All right Jim. Your quarterlies look very good. How are things at the library?	7.6	3706	2005-03-24
2	1	1	Pilot	Ken Kwapis	Ricky Gervais;Stephen Merchant;Greg Daniels	Jim	Oh, I told you. I couldn't close it. So...	Oh, I told you. I couldn't close it. So...	7.6	3706	2005-03-24
3	1	1	Pilot	Ken Kwapis	Ricky Gervais;Stephen Merchant;Greg Daniels	Michael	So you've come to the master for guidance? Is this what you're saying, grasshopper?	So you've come to the master for guidance? Is this what you're saying, grasshopper?	7.6	3706	2005-03-24
4	1	1	Pilot	Ken Kwapis	Ricky Gervais;Stephen Merchant;Greg Daniels	Jim	Actually, you called me in here, but yeah.	Actually, you called me in here, but yeah.	7.6	3706	2005-03-24
5	1	1	Pilot	Ken Kwapis	Ricky Gervais;Stephen Merchant;Greg Daniels	Michael	All right. Well, let me show you how it's done.	All right. Well, let me show you how it's done.	7.6	3706	2005-03-24
55130	9	24	Finale	Ken Kwapis	Greg Daniels	Dwight	No, don't say it. You're fired! You're both fired!	No, don't say it. You're fired! You're both fired!	9.7	7934	2013-05-16

Table 1: The Office dataset

season	number of episodes	average rating
1	6	8.0
2	22	8.4
3	23	8.6
4	14	8.6
5	26	8.5
6	24	8.2
7	24	8.3
8	24	7.7
9	23	8.0

Table 2: Number of episodes and average rating by season

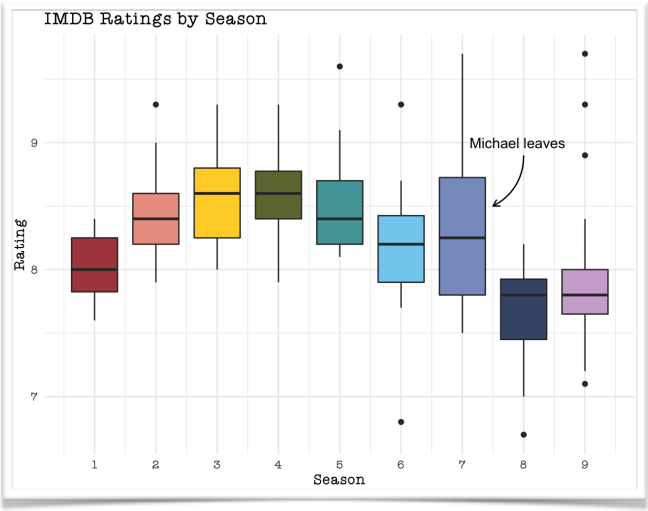

Figure 1: Distribution of ratings by season

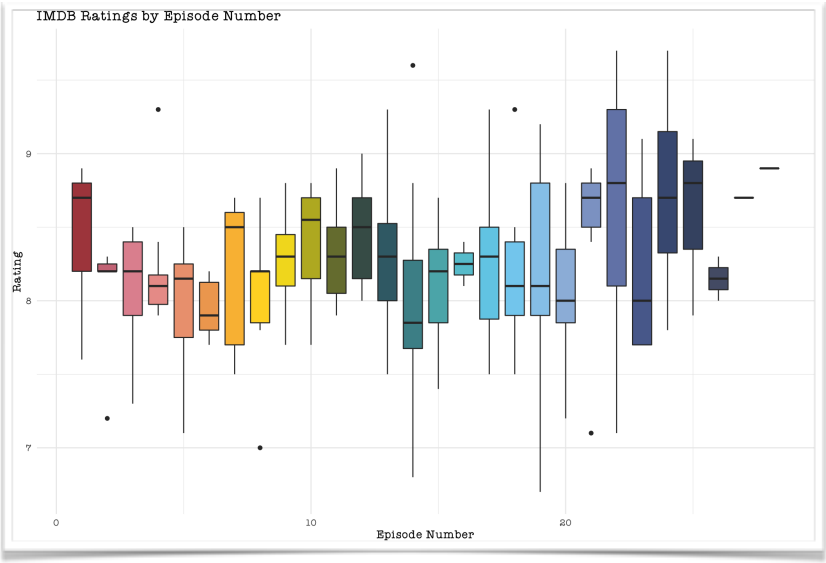

Figure 2: Distribution of ratings by episode

season	average total votes
1	3195
2	2631
3	2443
4	2423
5	2151
6	1856
7	2031
8	1546
9	1853

Table 3: Average number of total votes by season

Other than season, ratings also seem to show a trend based on the episode number in a season (Fig. 2). The seasons appear to start and end with higher median ratings with episodes in the middle of the season varying widely. The show increased in popularity in the first five seasons before tapering off (Fig. 3). It is also interesting to note that the total number of votes started declining gradually, with season 8 having the lowest average number of votes (Table 3). However, the last episode of the series, Finale, received the largest number of votes. The top 3 highest rated episodes come from seasons 5, 7, and 9 (Stress Relief, Goodbye Michael, Finale). Most of the lowest rated episodes occur in seasons 8 and 9 with one outlier, The Banker, positioned in season 6.

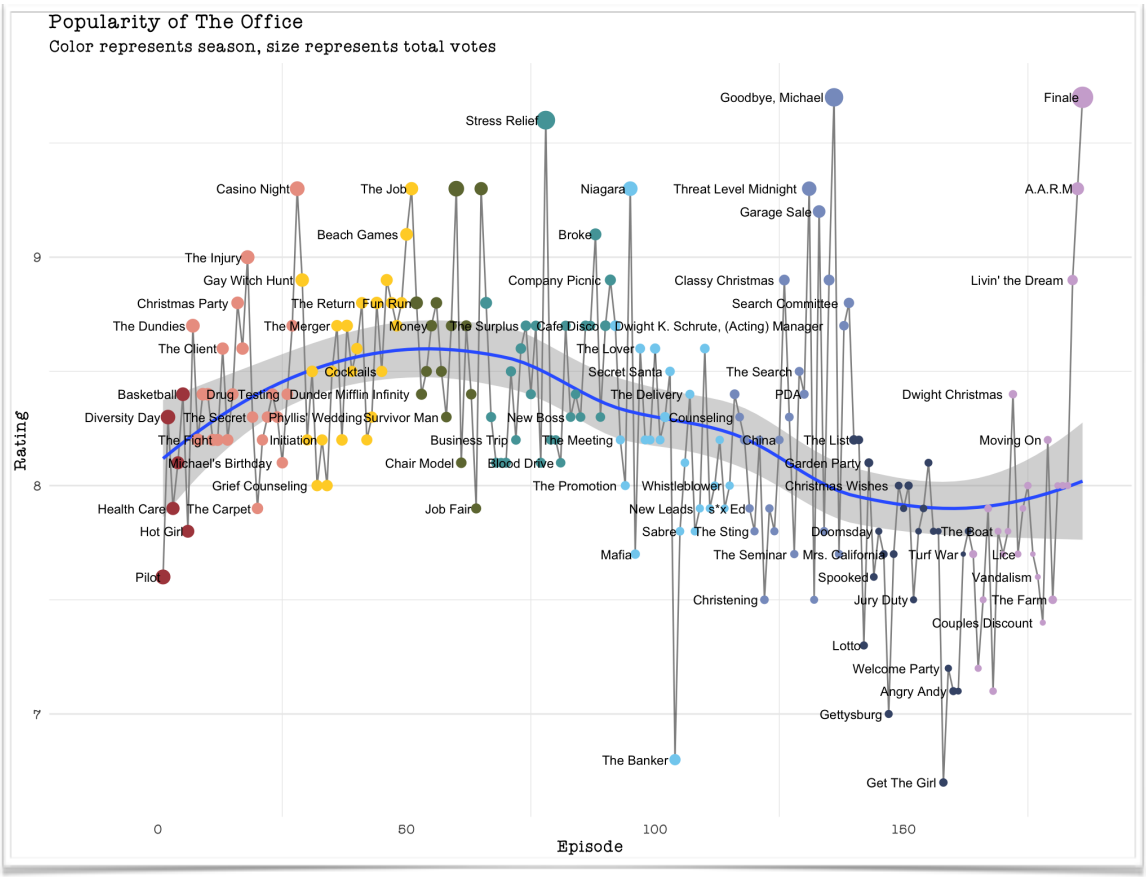

Figure 3: Rating for each episode throughout the series with a smoothing spline overlaid

A total of 55 directors were involved in the making of the show, many of whom directed only one or two episodes (Fig. 4). While only 10 directors were involved in more than five episodes (Table 4), most of the top 10 highest rated directors filmed between 2 to 4 episodes (Table 5), one of them being Steve Carell.

Top 10 Most Involved Directors		
director	No. Episodes	Average Rating
Greg Daniels	15	8.53
Randall Einhorn	15	8.17
Paul Feig	14	8.69
Ken Kwapis	12	8.54
Jeffrey Blitz	11	8.30
David Rogers	9	7.96
Ken Whittingham	9	8.33
Matt Sohn	8	7.85
Charles McDougall	7	8.31
Paul Lieberstein	7	8.14

Table 4: Number of episodes and average rating for the top 10 most involved directors

Top 10 Highest Rated Directors		
director	No. Episodes	Average Rating
Harold Ramis	4	8.75
Steve Carell	3	8.73
Jason Reitman	2	8.70
Paul Feig	14	8.69
Gene Stupnitsky	2	8.65
Joss Whedon	2	8.65
Lee Eisenberg	2	8.65
Tucker Gates	4	8.62
Julian Farino	2	8.60
Bryan Gordon	2	8.55

Table 5: Number of episodes and average rating for the top 10 highest rated directors

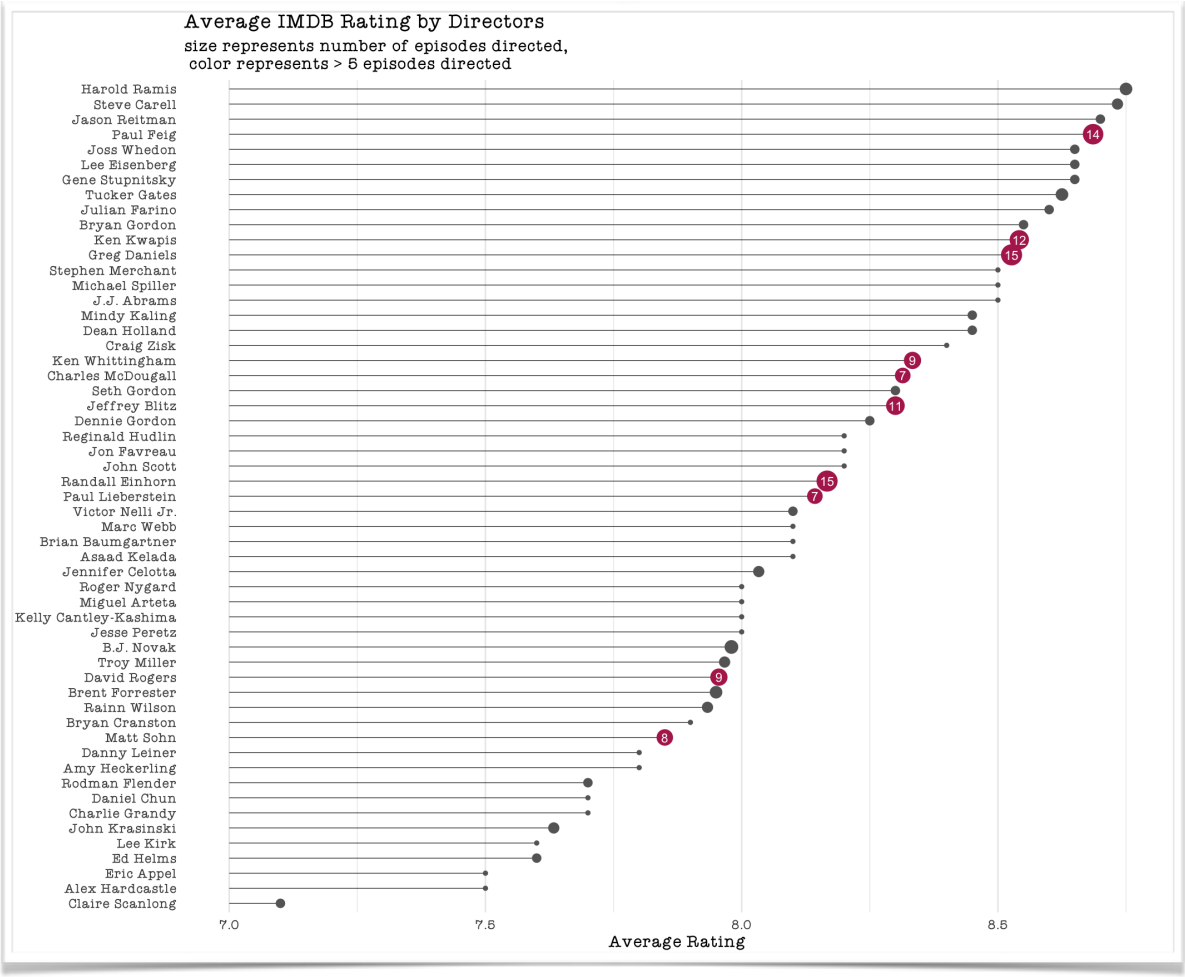

Figure 4: Average rating for each director. The colored points correspond to the directors in Table 4

Top 10 Most Involved Writers		
writer	No. Episodes	Average Rating
Mindy Kaling	21	8.39
Paul Lieberstein	16	8.54
B.J. Novak	15	8.37
Gene Stupnitsky	15	8.47
Lee Eisenberg	15	8.47
Greg Daniels	12	8.72
Brent Forrester	11	8.38
Jennifer Celotta	11	8.53
Justin Spitzer	11	8.27
Michael Schur	10	8.61

Table 6: Number of episodes and average rating for the top 10 most involved writers

Top 10 Highest Rated Writers		
writer	No. Episodes	Average Rating
Steve Carell	2	8.80
Greg Daniels	12	8.72
Michael Schur	10	8.61
Lester Lewis	2	8.60
Jon Vitti	2	8.55
Paul Lieberstein	16	8.54
Jennifer Celotta	11	8.53
Ryan Koh	2	8.50
Gene Stupnitsky	15	8.47
Lee Eisenberg	15	8.47

Table 7: Number of episodes and average rating for the top 10 highest rated writers

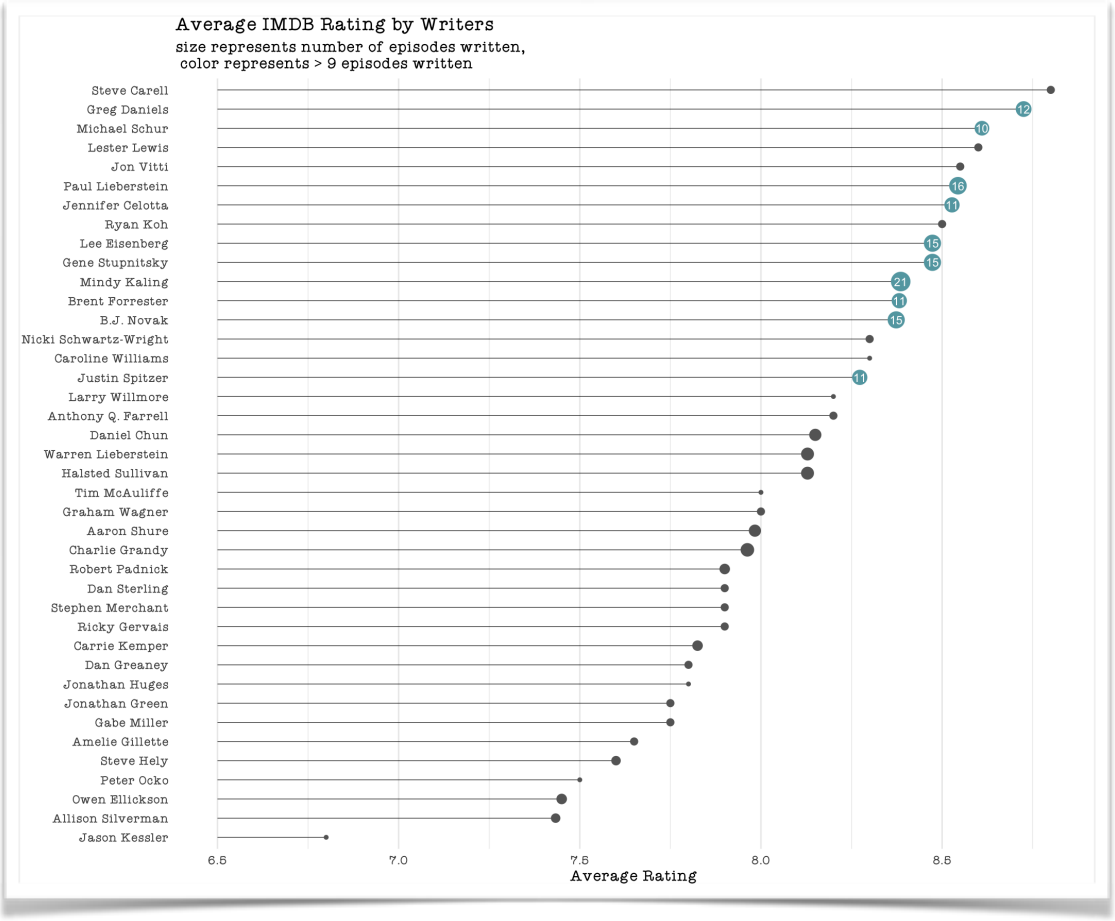

Figure 5: Average rating for each writer. The colored points correspond to the writers in Table 6

There were a total of 40 writers involved in the show (Fig. 5). Greg Daniels, who adapted the British version of the series for the American television, is one of the most involved writers (Table 6) and the second highest rated director (Table 7) after Steve Carell. It is interesting to see that some of the writers and directors involved in the show are also characters in the show — Mindy Kaling as Kelly, B.J. Novak as Ryan, Paul Lieberstein as Toby. Furthermore, the writer's role appears to have an influence in the ratings, more so than the director's role.

The Office has a large cast with several characters introduced late into the show or dismissed in the middle. After some minor data cleaning of characters' names, the dataset contains more than 700 unique characters. In order to determine the main characters, we look at the average proportion of lines per character (Fig. 6) and the proportion of episodes characters appeared in the show (Fig. 7). We consider main characters as those who had more than 100 total lines and appeared in more than two seasons. Michael, unsurprisingly, has the most lines in the show even though he leaves in season 7 and makes a last appearance in the Finale. Other characters pick up his share of lines, most notably Dwight, Pam, Andy. New characters are also introduced in the absence of Michael — Nellie, Robert, and Gabe. Dwight appeared in every single episode, followed by Jim, Pam, and Kevin.

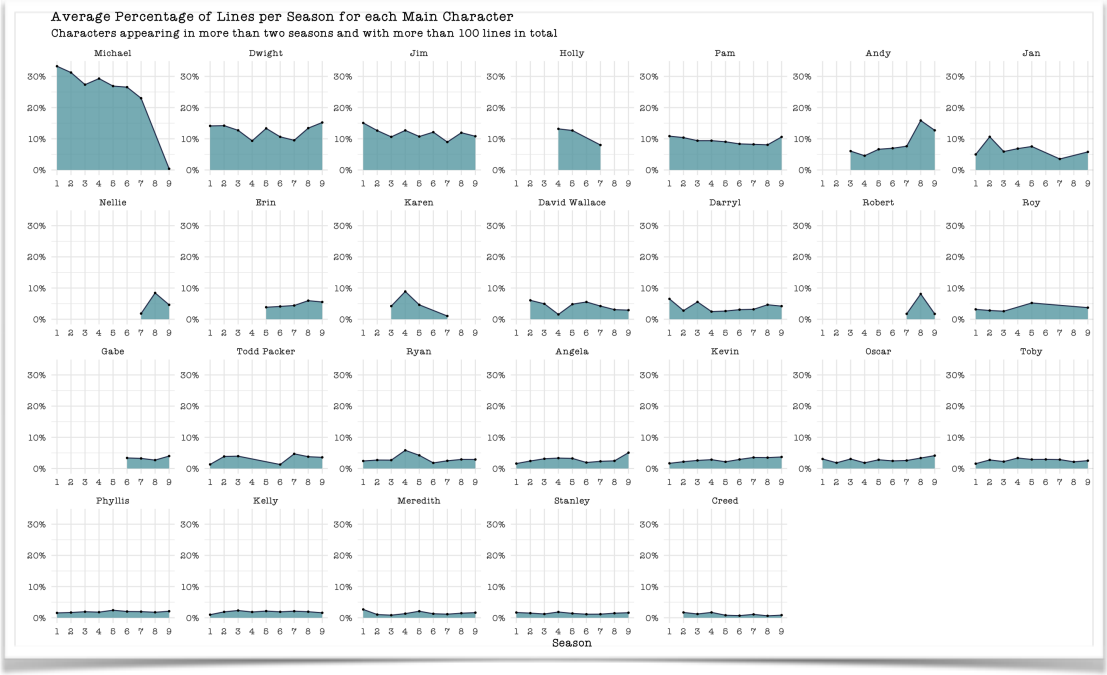

Figure 6: Average percentage of lines each season per main character

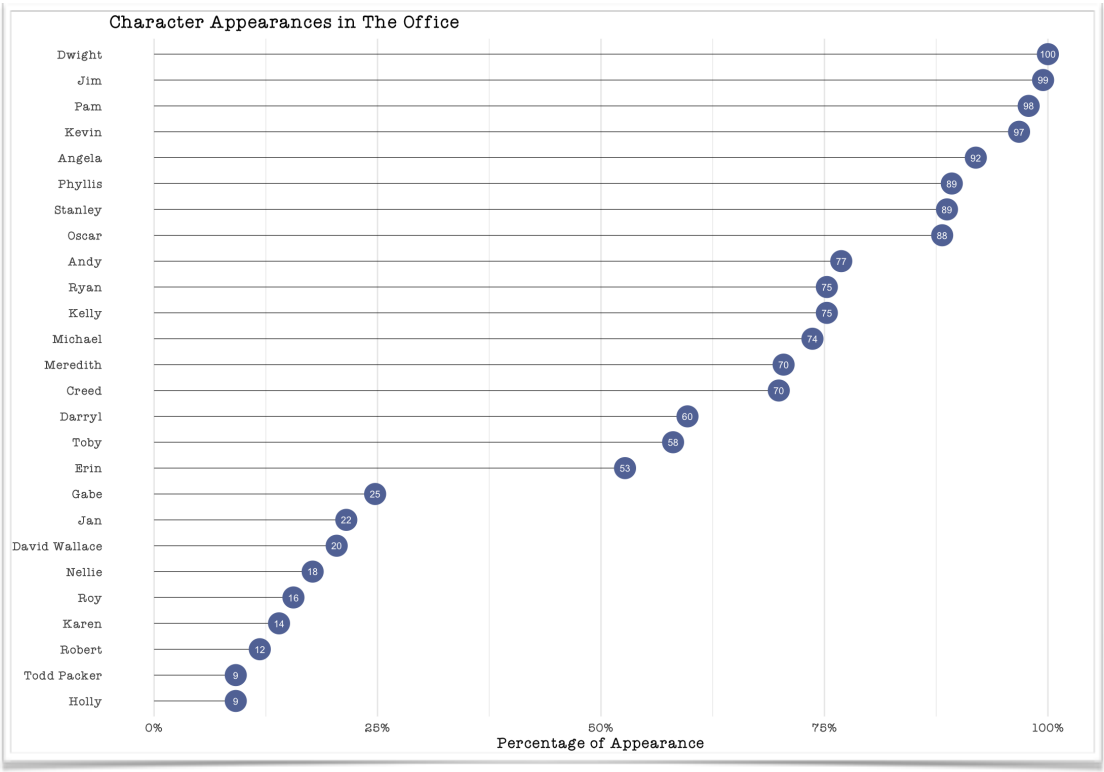

Figure 7: Percentage of episodes characters appeared in the series

2 METHOD

Unlike structured data, text data is considered unstructured. In order to represent text data as inputs for modeling, two steps are taken in text mining: preprocessing and feature engineering.

2.1 Preprocessing

In natural language processing, some of the common steps taken are tokenization, transforming text to lowercase, stop word removal, removing punctuations and numbers, and stemming or lemmatization […]. These steps are carefully planned depending on the task at hand due to the fact that they can affect the results. Preprocessing steps are dependent on the context of the data and can lead to different results. One reason to preprocess text data is to normalize text and reduce dimensionality. Textual data is inherently high dimensional and incredibly sparse, and these steps are taken to address these issues.

Tokenization involves splitting a stream of characters into discernible "tokens" in the form of a single character, word, n-grams, sentences, or even a whole paragraph [5, 24, 53]. The chosen method of tokenization can influence the results based on the tokenizer algorithm. For our clustering task, we use words as our tokens. For our regression task, we combine unigrams and bigrams as inputs to our model.

When the importance of proper nouns and capitalizations are trivial, transforming text into lowercase reduces the number of unique words, reducing computational cost. In addition, numeric strings may or may not hold information based on the context of the data. For example, financial text data might consider numeric characters valuable, whereas numbers in a fairytale might be meaningless. Another aspect to consider in natural language processing is whether punctuations carry information that might provide additional insights or patterns. We transform our text into lowercase for all other tasks and remove numbers and punctuations. For the regression task, although we dismiss punctuations and capitalizations, we include the counts of such occurrences as inputs to our model.

Stop words are words that carry very little to no information and become noise in our data, such as "the", "a", and "of". It is common practice to remove stop words to reduce the dimension of the text data and ease computational cost [24]. There are premade stop word lists available in R. However, stop words are also context-dependent, and premade lists might not transfer well to some text data. If a stop word list contains words that are highly informative within the context of the data at hand, we risk losing that information by removing valuable words from the text. Therefore, words should be removed judiciously. For this paper, we use the `Snowball` stop word list from the `stopwords` package with additional custom stop words appended based on initial evaluation of common words in our text data.

In the English language, words come in various tenses or modifications, such as plurals, adjectives, past tense, etc. By stemming, words are reduced to its roots. For example, "apples" are reduced to "appl" and "happily" to "happi". While stemming can greatly reduce the feature space of text data, it runs the risk of losing coherence or resulting in a root that has a complete different kind of meaning or semantic [24]. Furthermore, some research have shown stemming do not improve performance, and in some cases, can be detrimental to the performance, such as in topic modeling [24, 45]. Another approach to normalizing words and reducing the feature space is lemmatization. Unlike stemming, which is a rule-based approach, lemmatization utilizes the dictionary-based approach and relies heavily on linguistics. It identifies the word's meaning or role in the context and extracts its lemma based on a dictionary. For example, "happily" is identified as an adverb and its lemma is "happy". However, lemmatization requires more computational time and may not improve performance. For this paper, lemmatization is utilized for clustering and regression tasks through the `spacyr` and `textstem` packages in R.

2.2 Feature Engineering

After preprocessing text, the next step involves building features to represent text data as numeric inputs for statistical modeling. One of the most common methods for representing text in a numeric format is a "bag of words" model, where word order is not taken into account. It reduces dimensionality at the cost of losing semantic meaning. The most common representations are term frequencies, word associations, and term frequency-inverse document frequency (tf-idf), which is a statistic that measures how important or relevant a word is to a document in a corpus [5, 24]. It computes the frequency of a term in a document and weights it against how often the word occurs in the corpus. If a word is common throughout the corpus, like "the", the word is given a lower weight, whereas words that are common in a document but rare in the corpus are given a higher weight, thus resulting in a higher score. For our data, a document can be considered individual lines, all of the lines in one episode, or all of the lines in one season. A corpus is then a collection of individual lines, a collection of all episodes, or a collection of all seasons, depending on the task.

$$TF_{ij} = \frac{f_{ij}}{n_j}$$

where f_{ij} is the frequency of term i in document j. n_j is the total number of words in document j.

$$IDF_i = log(\frac{N}{df_i})$$

where N is the total number of documents in the corpus, df_i is the number of documents containing word i.

$$w_{ij} = TF_{ij} \times IDF_i$$

where w_{ij} is the TF-IDF score of term i in document j.

For example, we can explore the TF-IDF for each character (Fig. 8). The words most commonly spoken by Andy are "tuna", "nard", "bernard", and "cornell". Those who are familiar with the show can agree these are words specific to the character Andy. In fact, without labeling each group of words with character's name, it is relatively easy to guess who those group of words would belong to. By analyzing tf-idf of each character, we can also draw insights to the character's personality. We can also explore how word frequencies differ between seasons and throughout the entire series. Figure 9 shows how frequently words are used in each season compared to its frequency in the whole series. Words that are close to the line occur with similar frequencies across all seasons, while words further from the line appear more frequently in one season than another [49]. If words fall below the line, they are more common in that season but not throughout the series. On the other hand, if words stand out above the line, they are more common across the show, but not within that season. For example, "downsize" in season 1 happens more frequently in that season than throughout the show, whereas "scott", which is the last name of Michael, occurs less frequently in season 8 and 9 compared to the rest of the seasons. In addition, "dwight" occurs in similar frequency across most seasons since it falls on the line in 6 of the seasons, which is expected since he appears in every single episode based on Figure 7.

Another potentially useful features for text data are count features. These are simple statistics in linguistics that count the number of words or unique words, characters, digits, punctuations, capital letters, prepositions, first person, third person, etc. Depending on the data, some or all of these count features can provide additional textual information as inputs for the model. Another popular method is word embedding, which captures semantic meaning and is ideal for recurrent neural networks [11, 24,

34]. Words are represented in a vector space where they are relationally oriented and carry meaning in their positions [5]. For clustering and regression, we employ tf-idf method. We also use count features as additional inputs for regression.

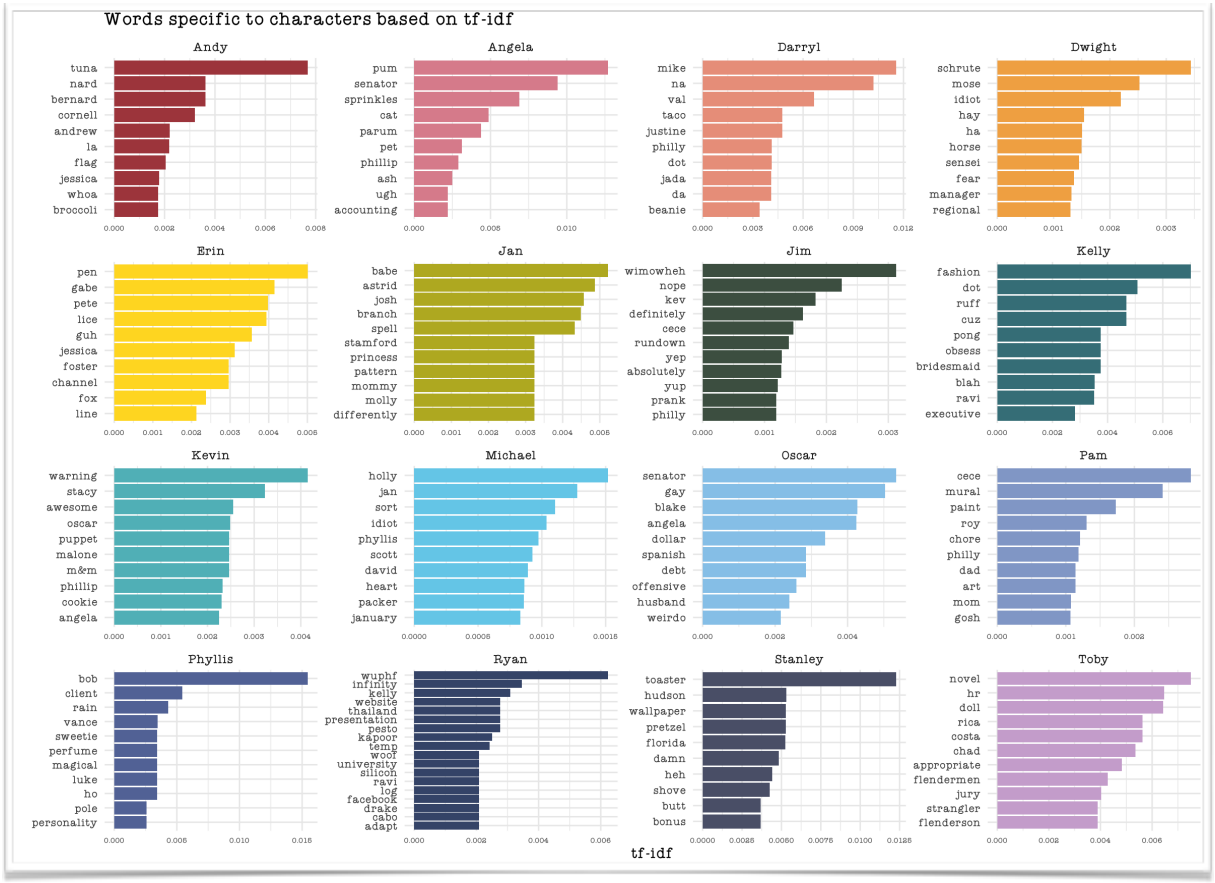

Figure 8: Top common words specific to top 16 characters based on tf-idf.
Note: main characters with less than 600 less not shown.

2.3 Sentiment Analysis

Sentiment analysis, also known as opinion mining, is the process of extracting emotional intent or opinions from text [24, 29, 34]. Sentiment can be a binary class (positive or negative), multi-class (very satisfied, satisfied, neutral, unsatisfied, very disappointing), or a continuous value. There are two main approaches to sentiment analysis. The machine learning approach uses labeled data to extract meaningful features from text and builds a classifier through various algorithms, such as Naive Bayes, SVM, and neural networks, to predict the sentiment of text data [55]. There are studies on sentiment classification using movie reviews, Twitter posts, customer feedbacks among countless others. However, the caveat is that the data needs to be labeled, which our data lacks.

Another approach is the lexicon-based approach which uses rules to identify the sentiment of text. By leveraging premade sentiment lexicons, words that have emotional intent are extracted from the text data and assigned a score or label. Then, the sentiment or polarity of the text is inferred by the combination of those words [47]. There are several lexicons available in R, such as AFINN, bing, nrc, Jockers, Sentiword, emoji, and internet slang. However, similar to stop word lists, lexicons are specific to the context of the text data since they were created and validated on datasets for a specific purpose. For

example, lexicons that do not take into account language from a different era will not transfer well to text data that contain either antiquated language style. In another example, if the text data comes from a younger demographic with high usage of slang where the word contains a meaning very different from the formal definition, a general-purpose lexicon can potentially assign sentiment incorrectly, such as the word “sick” in the context of illness as opposed to describing something as excellent. Therefore, lexicons also must be used with subjective reasoning.

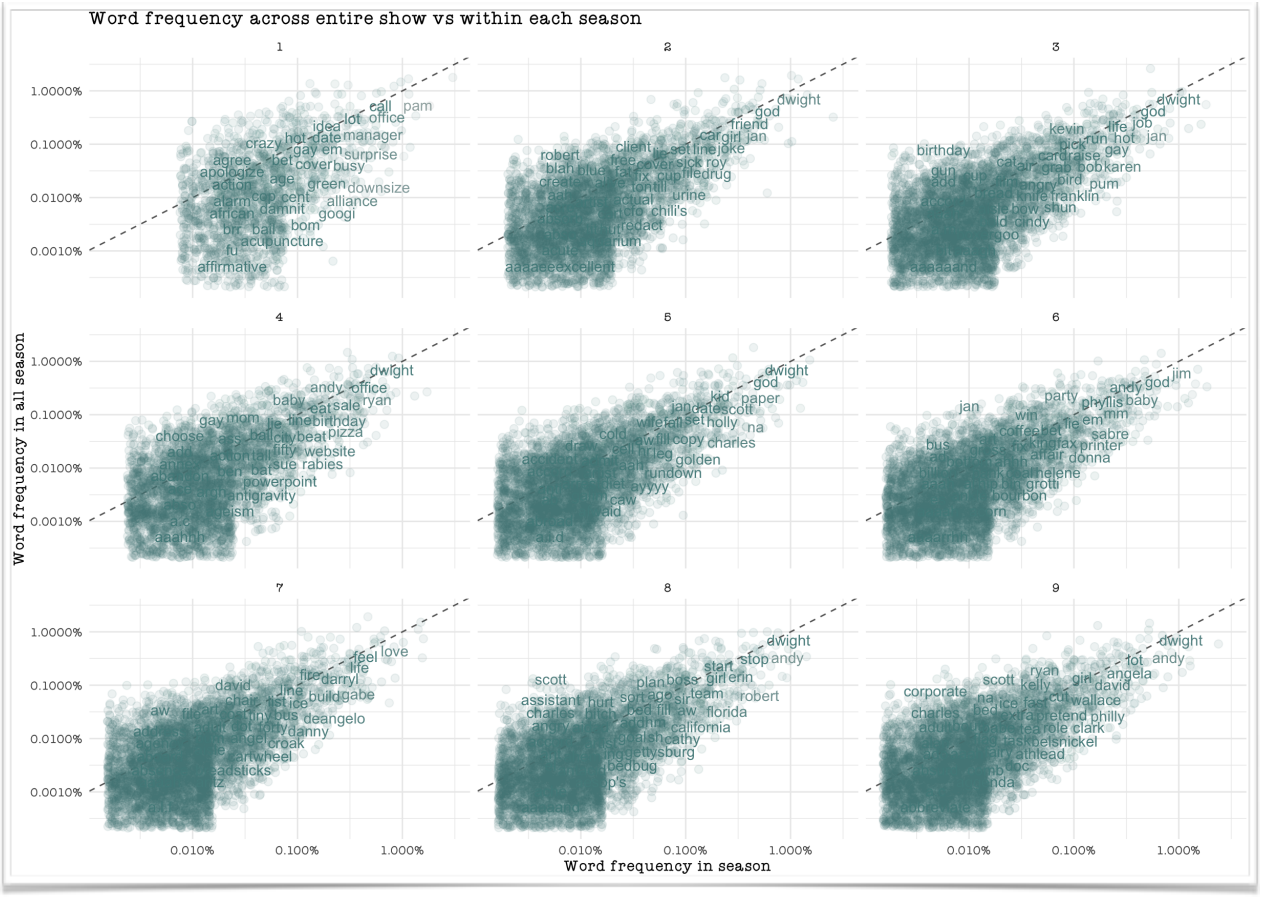

Figure 9: Word frequencies throughout all 9 seasons versus word frequencies within each season. Words that fall close to the line indicate similar frequencies across all seasons. Words that fall below the line are more

The Office is a show that is generally perceived as negative in sentiment by construction. For this paper, we analyze the sentiment of each main character using their lines as an exploratory analysis using Tyler Rinker’s `sentimentr` package [44]. The Sentiword lexicon assigns continuous values between -1 and 1. Therefore, we can compute the polarity of each main character throughout the series (Fig. 10). Stanley is, on average, the most negative character, followed by Angela, Phyllis, and Dwight. Surprisingly, the most negative season for some of the characters was season 1, which had only 6 episodes. Dwight and Jim, who appeared the most in the series, show consistent sentiment scoring for all seasons with Jim being the most positive character on average, followed by Michael. Furthermore, Michael who appeared only once for the finale in season 9 was the most positive for that season. However, the overall sentiment of these characters were negative, which was expected.

We also use NRC's lexicon to explore the more nuanced emotions of these characters, which assigns two types of categorical values: binary (positive/negative) and multi-class (8 emotions). In Figure 11, we can see how characters cluster based on sentiment. Stanley, Darryl, and Andy loosely cluster together. Stanley displays anger, sadness, and disgust the most, while Darryl expresses more sadness and Andy shows more anger than any other emotions. Toby and Oscar seem to cluster together since they both express surprise and trust the most. Jim, Pam, and Ryan also seem to create a cluster of their own for they display anticipation the most and sadness, disgust, and anger the least. Lastly, we can see the top 100 positive and negative words spoken by these characters as a whole (Fig. 12).

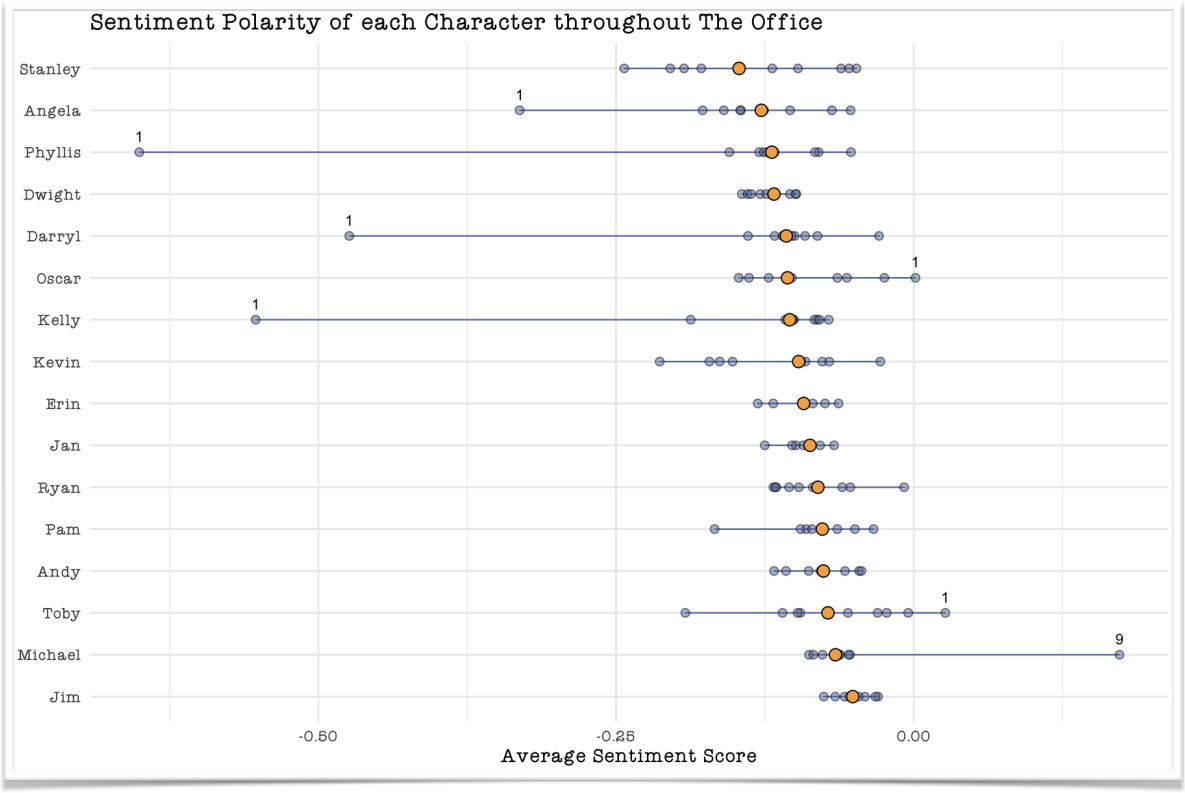

Figure 10: Average sentiment score for each main character for each season in the show based on Sentiword lexicon. The yellow points indicate the overall average sentiment score. Note: main characters with less than 600 less not shown.

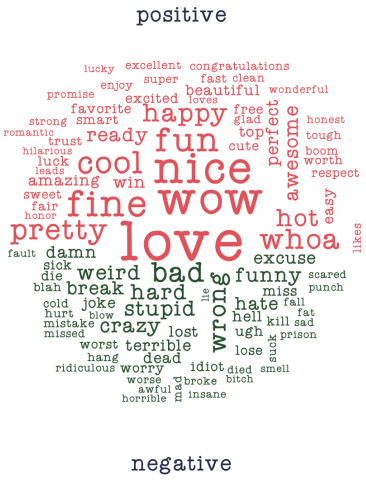

Figure 12: Wordcloud of top 100 positive and negative words spoken by main characters

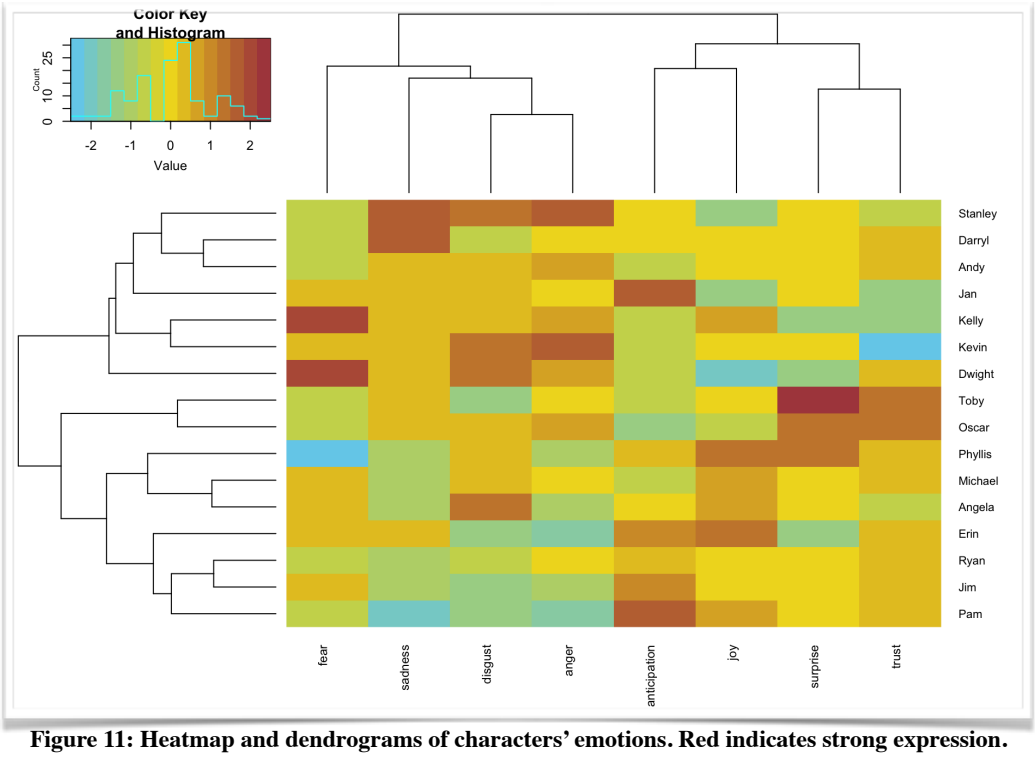

Figure 11: Heatmap and dendrograms of characters' emotions. Red indicates strong expression. Blue represents weak expression.

2.4 Clustering

Text clustering has been widely used and studied in various domains [4, 5, 21]. The idea is to cluster documents using either similarity or difference measure based on textual data [29]. While it can be used as a tool for dimensionality reduction, it is also useful as an exploratory tool. For this paper, we use hierarchical agglomerative clustering algorithm to explore the transcripts of the main characters to extract meaningful clusters of characters. Hierarchical dendrograms provide a relatively easy visualization for clustering text and interpretation [29]. Huang details different types of distance and similarity measures for clustering text document, such as the Euclidean distance and cosine similarity [21]. For text documents, cosine similarity is widely used. Given two documents A and B, where they are n-dimensional vectors, the cosine similarity is

$$C = \frac{\mathrm{A} \cdot \mathrm{B}}{||\mathrm{A}|| ||\mathrm{B}||}$$

and it is bounded between 0 and 1. One of its properties is that it is independent of document length. The `proxy` package provides different distance and similarity measures.

2.5 Text Regression

The goal is to predict the IMDB rating using features available in our data. In order to use text data as inputs to our model, we perform preprocessing as previously mentioned. Then, we extract tokens to compute tf-idf. Per Google's recommendation, we utilize both unigram and bigram as tokens for the model [2]. We also leverage count features, such as number of words and number of capitalizations, using the `textfeatures` package. With these new features, we combine non-text data: season, episode number,

director, writer, number of lines per main character, and total number of votes. Before modeling, we also remove variables with zero variance and large absolute correlations with other variables. This results in a total of 1,121 features and 186 episodes. After several initial runs, limiting the number of tokens to a maximum of 1,000 resulted in faster and better performance for all the models, and increasing it led to negligible changes or decreased performance. Thus, we set the number of tokens to 1,000.

We employ lasso regression, support vector regression (SVR), gradient boosting (XGBoost), k-nearest neighbors (kNN), and neural network models. With the exception of XGBoost, the feature space was normalized. In order to evaluate the performance of the models, we split the data into 80% training data and 20% test data. Furthermore, we use 10-fold cross-validation on the training data to tune the hyperparameters for each model using the smallest root mean squared error (RMSE). We also look at mean absolute error (MAE) and mean absolute percentage error (MAPE). Let $e_i = y_i - \hat{y}_i$. Then,

$$RMSE = \sqrt{\frac{1}{n}\sum e_i^2}$$

$$MAE = \frac{1}{n}\sum |e_i|$$

$$MAPE = \frac{100}{n}\sum \frac{|e_i|}{y_i}$$

Once the best hyperparameters are found for each set of models, we fit the final best model to all of the training dataset in order to evaluate the performance of each model on the unseen test data.

3 RESULT

3.1 Clustering

The goal is to see if we can extract any natural groups within our main characters using hierarchical agglomerative clustering. After experimenting with different distance and similarity metrics and number of clusters, the most meaningful clusters resulted using the cosine similarity metric and Ward's clustering algorithm [21, 38]. The main characters are clustered into 10 groups based on the similarity of their lines in the show (Fig. 13). Based on the characters' plots in The Office, these groups surprisingly make sense. While a few of the groups can be slightly more difficult to interpret, most of the clusters can be clearly labeled. In the first cluster, Kelly and Ryan were romantically involved. Oscar, Angela, and Kevin, who are grouped in the second cluster, made up the accounting department. David Wallace and Jan were part of the corporate team. Holly and Toby both worked in human resources. Karen, Phyllis, and Stanley were part of the sales team and sat closely together in the office. The biggest cluster is made up of Darryl, Andy, Pam, Jim, Dwight, and Michael. With the exception of Pam, all of these characters were either a manager or assistant (to the) manager at one point in the show. Pam was the receptionist and had many close interactions with Michael and Dwight. She was also married to Jim.

3.2 Text Regression

For tuning hyperparameters, we used grid search for lasso, which was computationally fast relative to other models. Figure 14 shows how lasso performed across various regularization penalties. Furthermore, the lasso model reduced the number of features from 1,126 to 68. Variables considered important to the lasso model is shown in Figure 15, while non-text variables considered important to the lasso model is displayed in Figure 16.

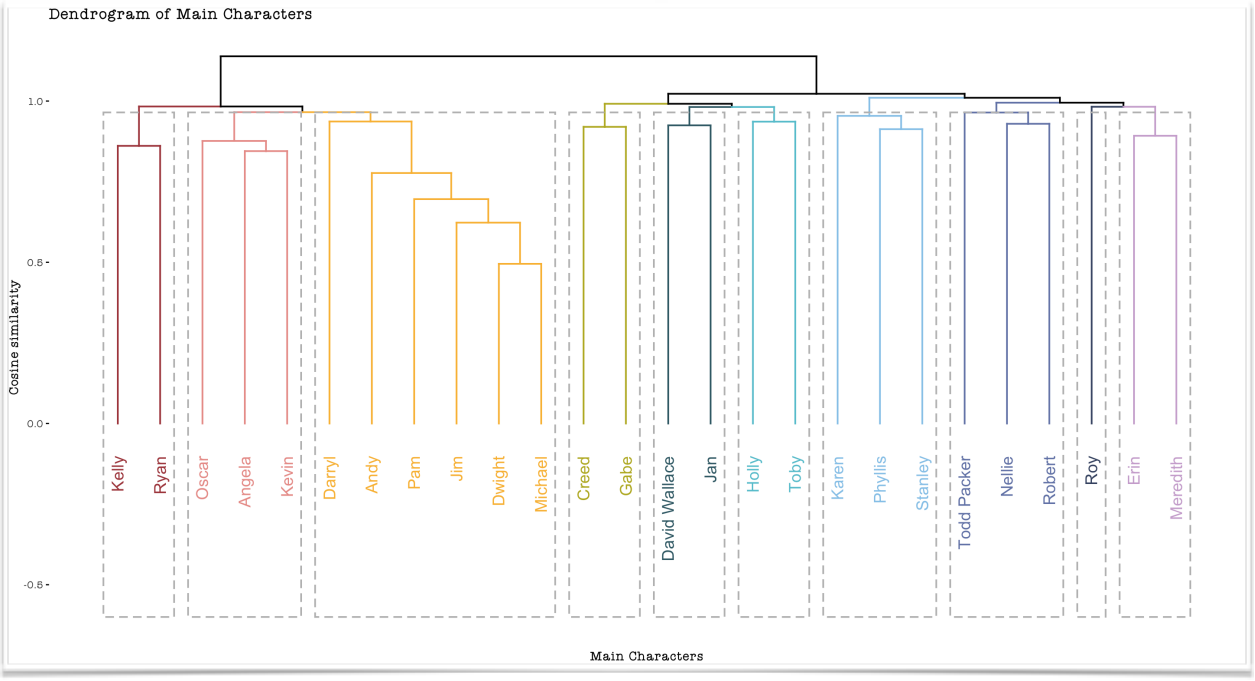

Figure 13: Hierarchical clustering of main characters using cosine similarity and Ward's algorithm with k=10

Support vector regression has four hyperparameters that need to be tuned, which are cost, margin, polynomial degree, and scale factor. After cross-validation, we can see how this model performed across different combinations of the hyperparameters (Fig. 17). The best model was chosen based on RMSE. For XGBoost model, there are also four hyperparameters, which are number of trees, number of predictors that will be randomly sampled at each split when creating the tree models (mtry), the maximum depth of the tree, and the learning rate from interation to iteration. The XGBoost model's performance across different hyperparameters are shown in (Fig. 18). The standard error for each hyperparameter is plotted, however, due to its small value it does not show clearly in the plot. We can also take a look at the variables considered important to XGBoost model (Fig. 19 & 20). Both lasso and XGBoost models consider total number of votes as the most important variable. We can also see Michael, Nellie, and Dwight in both variable importance plots, as well as season 8. Unlike XGBoost, lasso did not consider count features as important predictors. In addition, writers and directors seem to be missing from the variable importance plot for XGBoost. K-nearest neighbors performance is also shown in Figure 21. The performance of the models based on the 10-fold cross validation is shown in Figure 22. Lasso and XGBoost seem to outperform the rest of the models. Neural net performed the worst, but this can perhaps be due to small dataset, since we only have 186 episodes in total.

Using the best hyperparameters, we fit the full training set and evaluate the performance of the models on the test data set, with the exception of neural network model since it performed the worst (Table 8). Lasso was the best model followed by XGBoost, however, looking at Figure 23, we don't see great results.

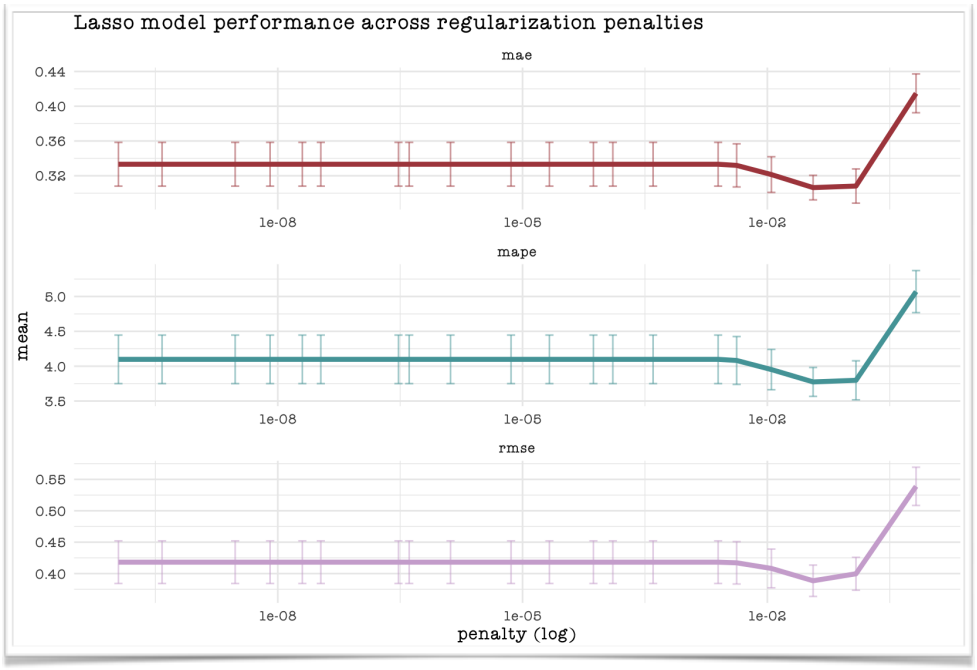

Figure 14: Performance of lasso regression for different penalty values based on MAE, MAPE, and RMSE with +/- 1 standard error.

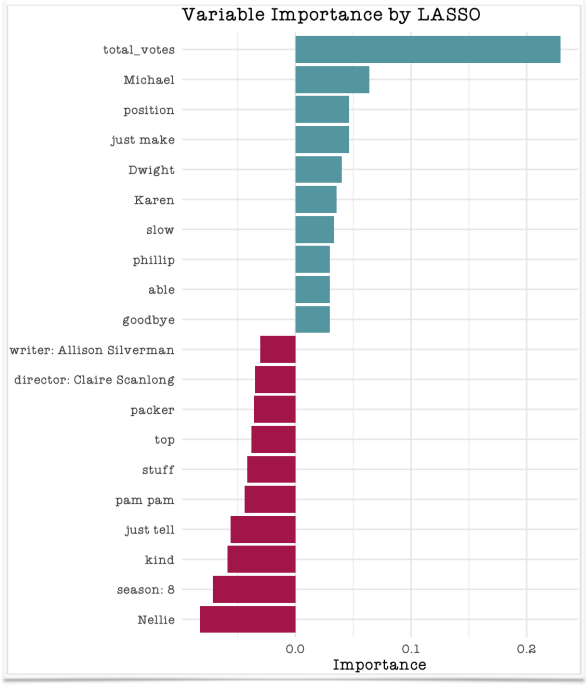

Figure 15: Variable importance in lasso model

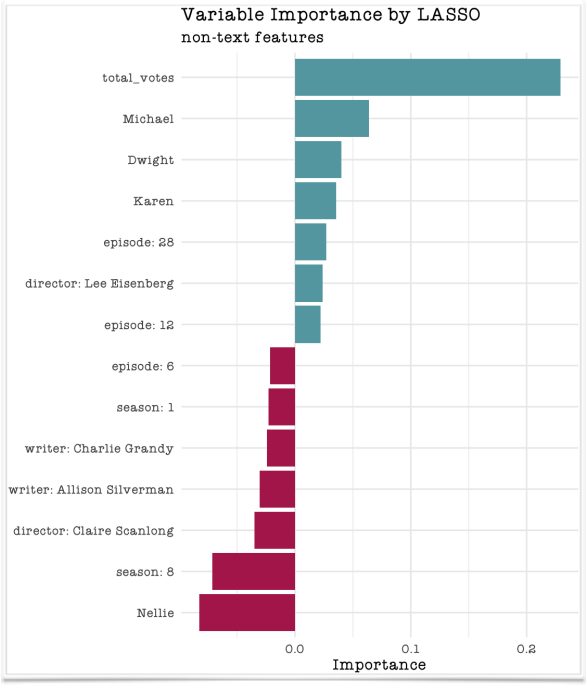

Figure 16: Variable importance in lasso model without text features

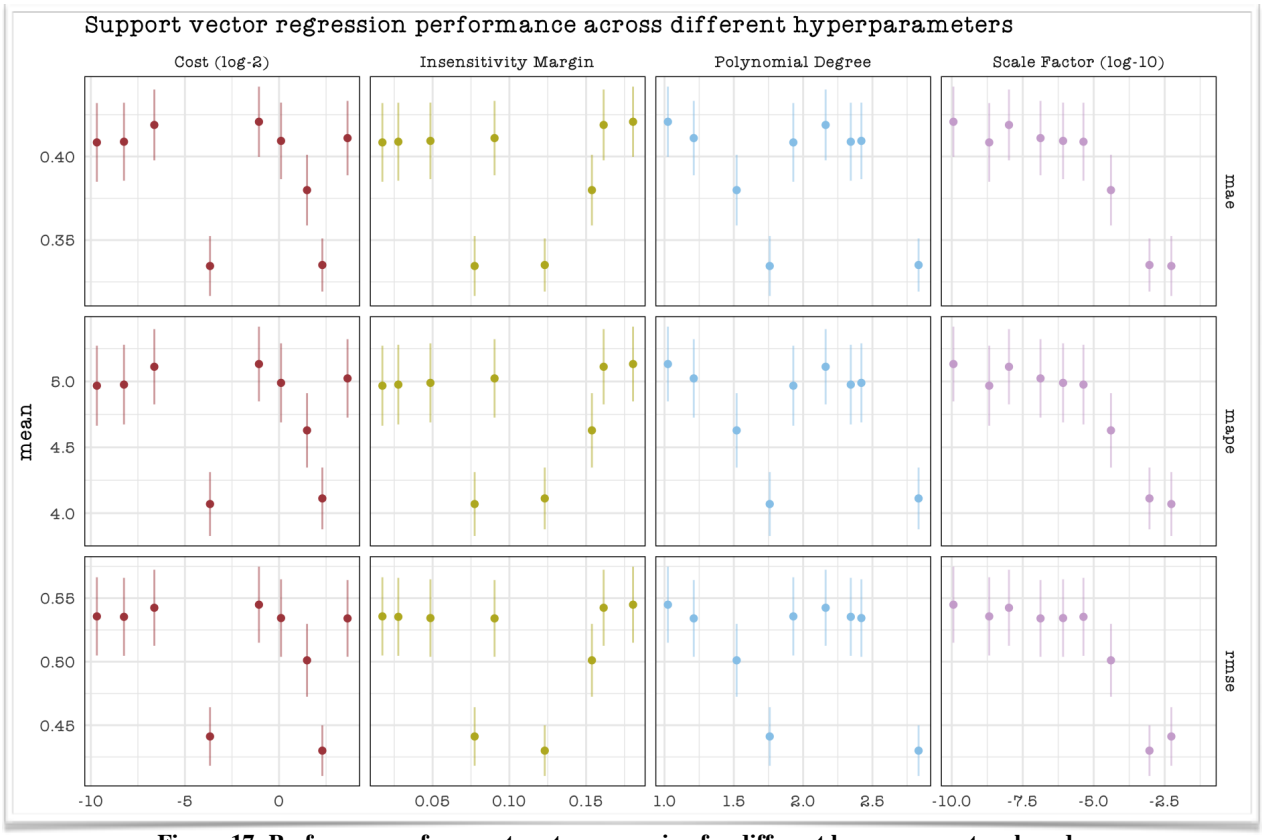

Figure 17: Performance of support vector regression for different hyperparameters based on MAE, MAPE, and RMSE with +/- 1 standard error.

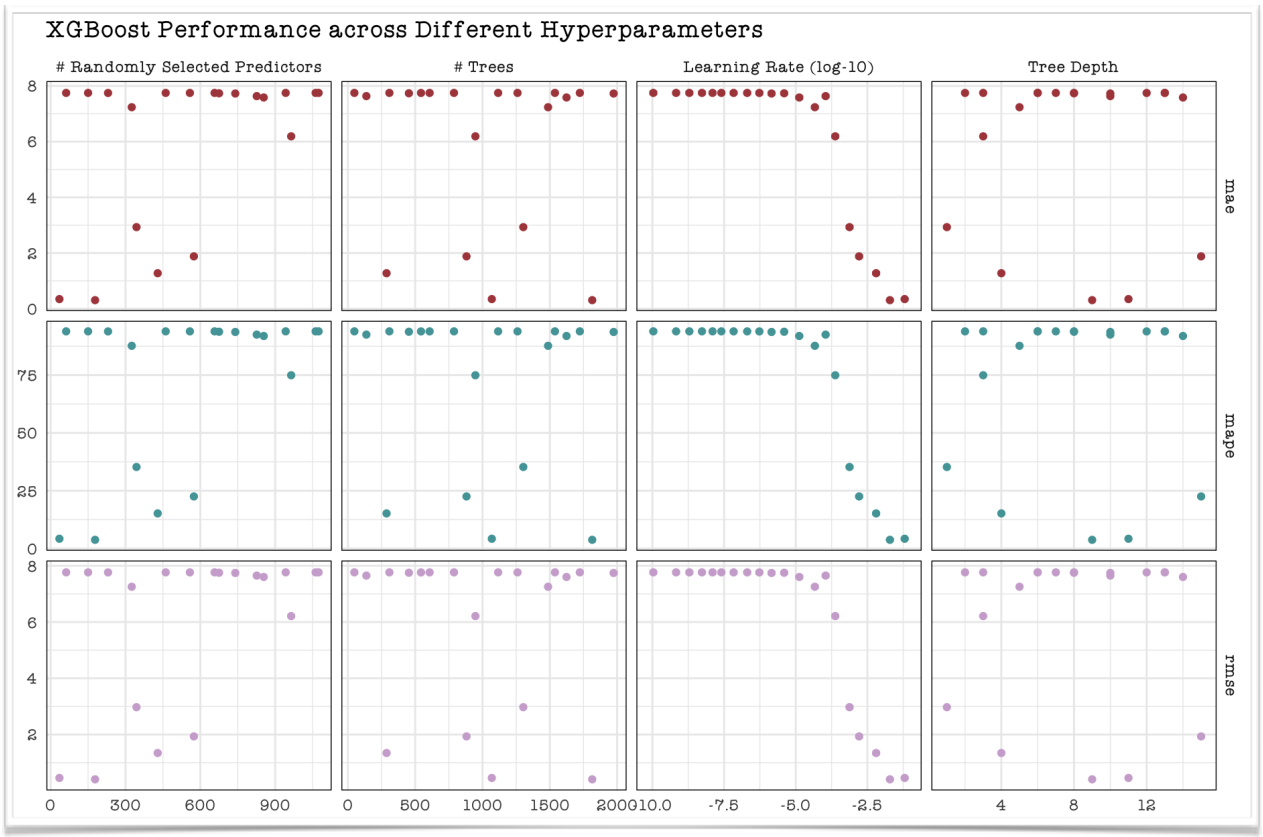

Figure 18: Performance of XGBoost for different hyperparameters based on MAE, MAPE, and RMSE with +/- 1 standard error.

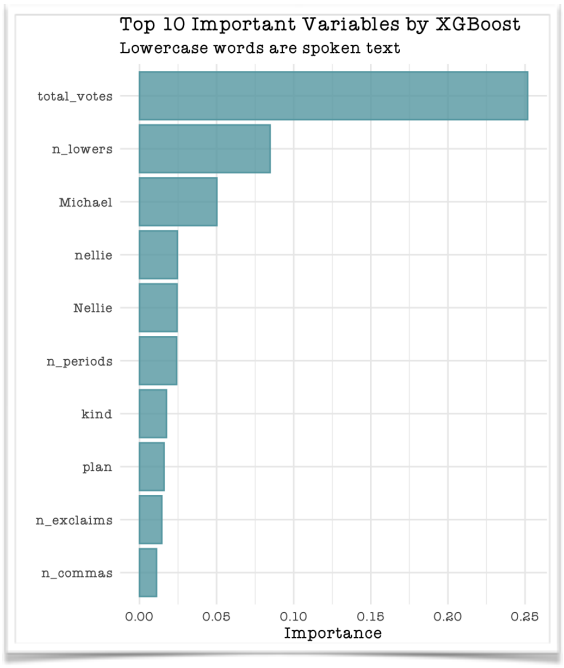

Figure 19: Variable importance in XGBoost model

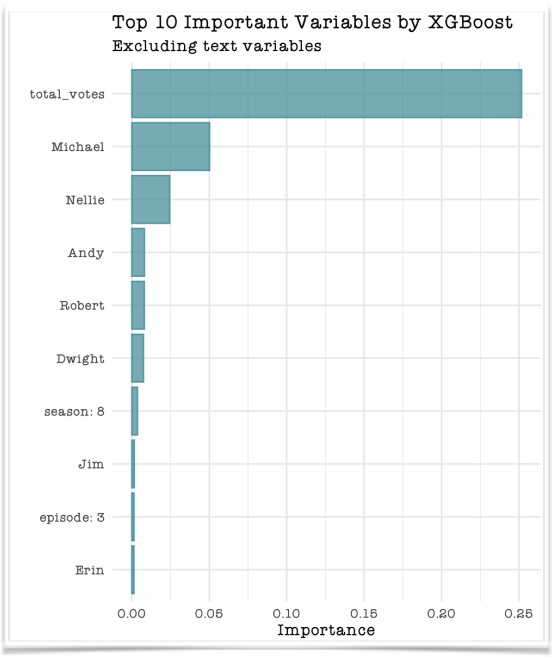

Figure 20: Variable importance in XGBoost model without text features

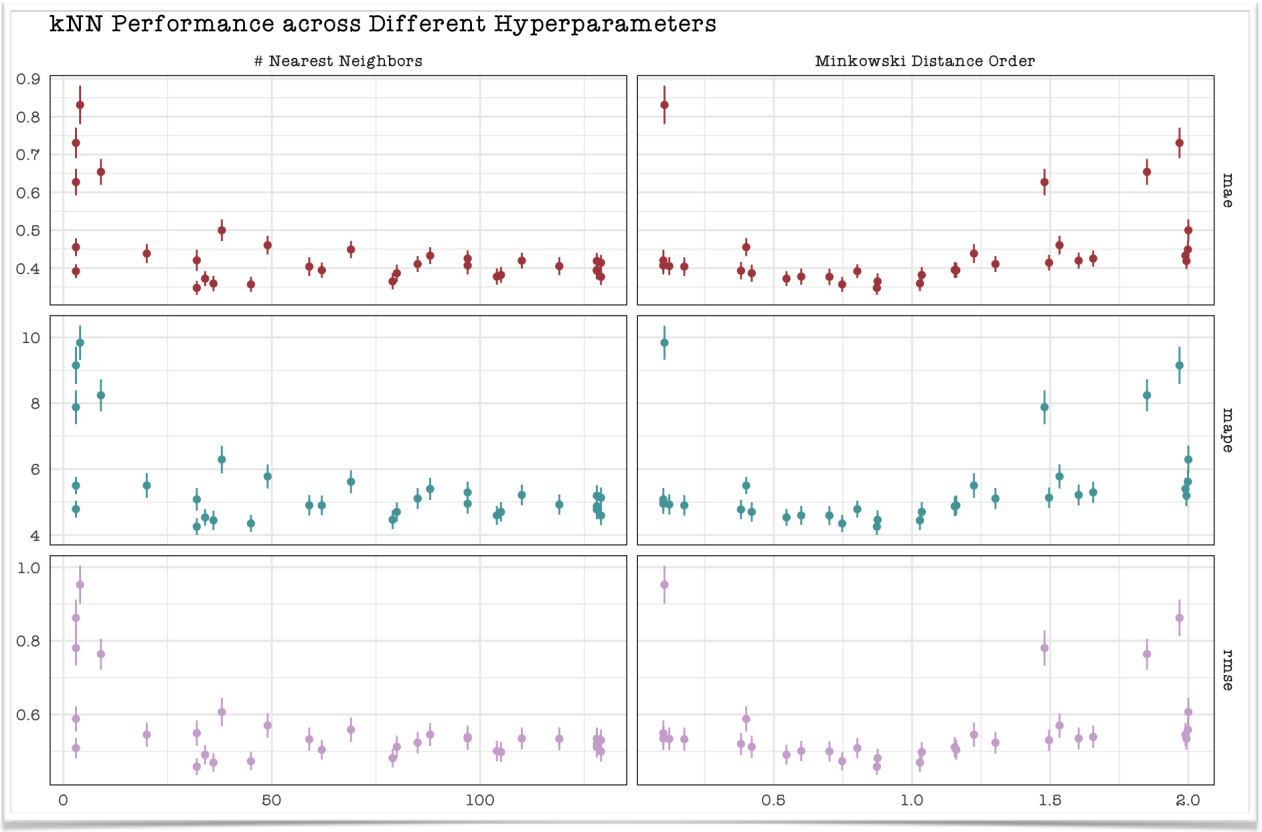

Figure 21: Performance of kNN for different hyperparameters based on MAE, MAPE, and RMSE with +/- 1 standard error.

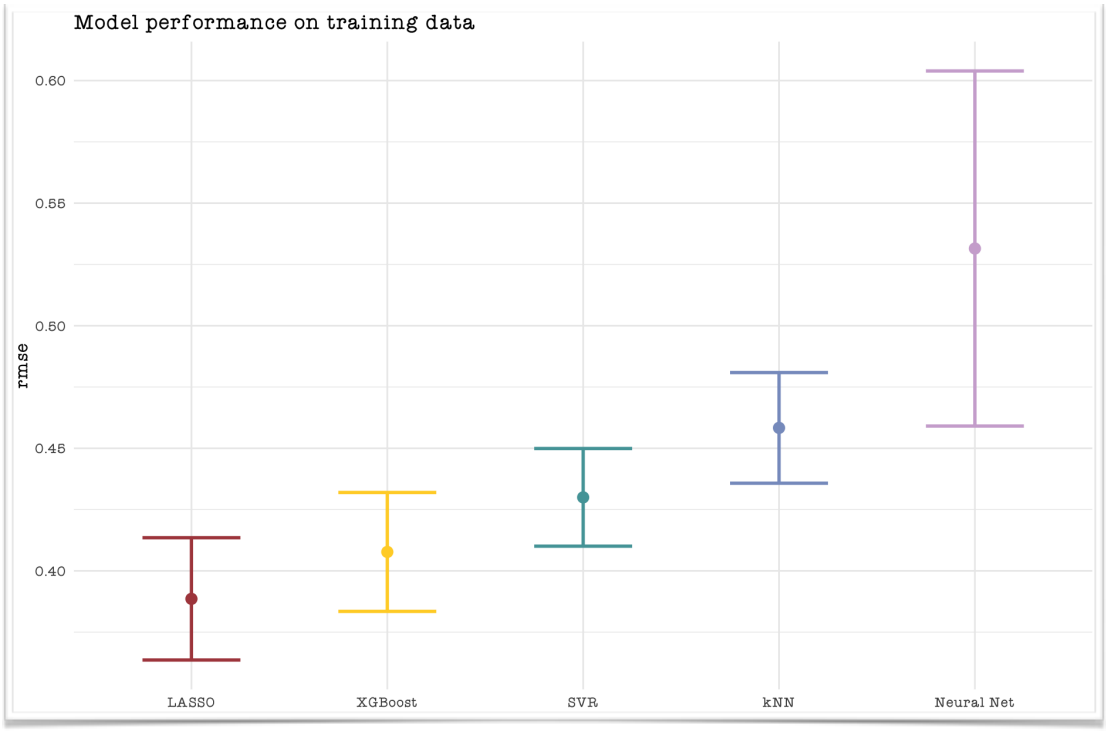

Figure 22: Comparison of model performance based on RMSE with +/- 1 standard error.

model	rmse	rsq
LASSO	0.3609198	0.4928825
SVR	0.4195354	0.2527381
XGBoost	0.3623126	0.4844341
kNN	0.3927961	0.3722316

Table 8: Final evaluation on test data

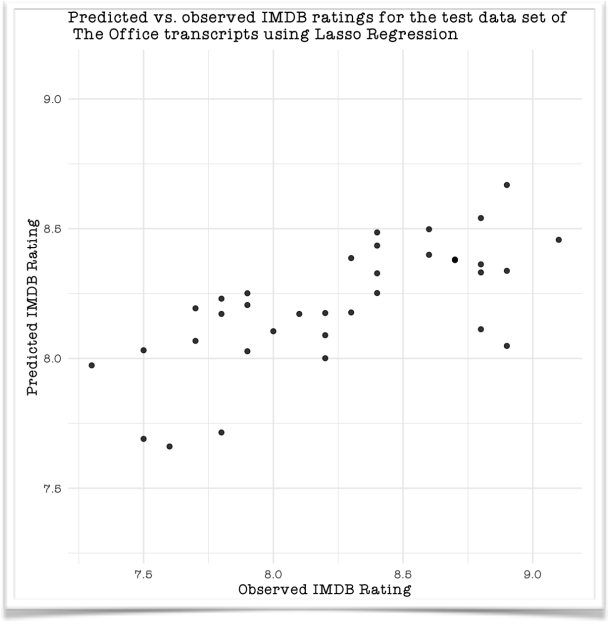

Figure 23: Lasso model predictions on test data.

4 CONCLUSION & FUTURE WORK

Using the transcript of a television show, The Office, we showed how text mining can provide meaningful insights to text data. We performed exploratory analysis, such as sentiment analysis and clustering, using lines of the main characters of the series. Although the text regression did not return impeccable results, we can see how text can be valuable as inputs to any regression or classification task, as well as unsupervised learning. Instead of considering rating as a continuous value, it might be worth looking into discretizing the output and perform classification. Another idea is to do more preprocessing and feature engineering as well as combining information extracted from unsupervised learning as another predictor.

This paper tapped only just a tip of the iceberg with regards to natural language processing and text mining. There are many tasks and methods that can be applied to this dataset, such as text classification, authorship attribution, text summarization, character network analysis, and topic modeling. Furthermore, we have not considered a richer representation of text, such as word embedding. In the future, applying deep neural network, such as CNN or RNN, with word embedding could potentially be a better method for text regression. Text generation was initially experimented with for this paper, which has not been included due to time and computational constraints, but it is a growing subfield of research and will be a valuable method to learn and practice.

While this dataset was exciting to explore and apply text mining, there are many other text data that can be analyzed using methods discussed in this paper and much more, especially deep learning. Due to the growing volume of text data, it is important to have tools and methods for text mining in a data scientist's arsenal.

5 REFERENCES

[1] URL: https://www.nbc.com/the-office.

[2] URL: https://developers.google.com/machine-learning/guides/text-classification/step-3.

[3] Hammad Afzal. "Prediction of Movies popularity Using Machine Learning Techniques". In: *International Journal of Computer Science and Network Security,* 16 (Aug. 2016), pp. 127–131.

[4] Charu C. Aggarwal and ChengXiang Zhai. "A Survey of Text Clustering Algorithms". In: *Mining Text Data*. Ed. by Charu C. Aggarwal and ChengXiang Zhai. Boston, MA: Springer US, 2012, pp. 77–128. ISBN: 978-1-4614-3223-4. DOI: 10.1007/978-1-4614-3223-4_4. URL: https://doi.org/10.1007/978-1-4614-3223-4_4.

[5] Mehdi Allahyari et al. *A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques*. 2017. arXiv: 1707.02919 [cs.CL].

[6] Varshit Battu et al. "Predicting the Genre and Rating of a Movie Based on its Synopsis". In: *PACLIC*. 2018.

[7] Zsolt Bitvai and Trevor Cohn. "Non-Linear Text Regression with a Deep Convolutional Neural Network". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 180–185. DOI: 10.3115/v1/P15-2030. URL: https://www.aclweb.org/anthology/P15-2030.

[8] D. Blei and J. Lafferty. "Dynamic topic models". In: *ICML '06*. 2006.

[9] Jonathan Chang, Jordan Boyd-Graber, and David M. Blei. "Connections between the Lines: Augmenting Social Networks with Text". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '09. Paris, France: Association for Computing Machinery, 2009, pp. 169–178. ISBN: 9781605584959. DOI: 10.1145/1557019.1557044. URL: https://doi-org.lib-proxy.fullerton.edu/10.1145/1557019.1557044.

[10] *COVID-19 Open Research Dataset Challenge (CORD-19)*. URL: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge.

[11] R. Das, M. Zaheer, and Chris Dyer. "Gaussian LDA for Topic Models with Word Embeddings". In: *ACL*. 2015.

[12] Nesat Dereli and M. Saraçlar. "Convolutional Neural Networks for Financial Text Regression". In: *ACL*. 2019.

[13] Mahidhar Dwarampudi and Dr Reddy. "Twitter Sentiment Analysis using Distributed Word and Sentence Representation". In: (Apr. 2019).

[14] L. A. Francis. "Taming Text: An Introduction to Text Mining". In: 2006.

[15] Paschalis Frangidis, Konstantinos Georgiou, and Stefanos Papadopoulos. "Sentiment Analysis on Movie Scripts and Reviews". In: *Artificial Intelligence Applications and Innovations* 583 (2020), pp. 430–438.

[16] Thushan Ganegedara. *Natural Language Processing with TensorFlow: Teach Language to Machines Using Python's Deep Learning Library*. Packt Publishing, 2018. ISBN: 1788478312.

[17] Matthew Gentzkow, Bryan Kelly, and Matt Taddy. "Text as Data". In: *Journal of Economic Literature* 57.3 (Sept. 2019), pp. 535–74. DOI: 10.1257/jel.20181020. URL: https://www.aeaweb.org/articles?id=10.1257/jel.20181020.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[19] T. Hashimoto, Hugh Zhang, and Percy Liang. "Unifying Human and Statistical Evaluation for Natural Language Generation". In: *NAACL-HLT*. 2019.

[20] G. Heinrich. "A Generic Approach to Topic Models". In: *ECML/PKDD*. 2009.

[21] Anna Huang. "Similarity measures for text document clustering". In: *Proceedings of the 6th New Zealand Computer Science Research Student Conference* (Jan. 2008).

[22] S. Hunter et al. “Moneyball for TV: A Model for Forecasting the Audience of New Dramatic Television Series”. In: *Studies in Media and Communication* 4 (2016), pp. 13–22.

[23] S. D. Hunter, Susan Smith, and Ravi Chinta. “Predicting New TV Series Ratings from their Pilot Episode Scripts”. In: *International Journal of English Linguistics* 6 (2016), p. 1.

[24] Emil Hvitfeldt and Julia Silge. *Supervised Machine Learning for Text Analysis in R*. URL: https://smltar.com/.

[25] Chowdhury Md Intisar et al. “Classification of Programming Problems Based on Topic Modeling”. In: *Proceedings of the 2019 7th International Conference on Information and Education Technology*. ICIET 2019. Aizu-Wakamatsu, Japan: Association for Computing Machinery, 2019, pp. 275–283. ISBN: 9781450366397. DOI: 10.1145/3323771.3323795. URL: https://doi.org/10.1145/3323771.3323795.

[26] Mahesh Joshi et al. “Movie Reviews and Revenues: An Experiment in Text Regression.” In: June 2010, pp. 293–296.

[27] Prasad Kawthekar. “Evaluating Generative Models for Text Generation”. In: 2017.

[28] V. Kobayashi et al. “Text Mining in Organizational Research”. In: *Organizational Research Methods* 21 (2018), pp. 733–765.

[29] Ted Kwartler. “Predictive Modeling: Using Text for Classifying and Predicting Outcomes”. In: *Text Mining in Practice with R*. John Wiley & Sons, Ltd, 2017. Chap. 7, pp. 209–236. ISBN: 9781119282105. DOI: https://doi.org/10.1002/9781119282105.ch7. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119282105.ch7. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119282105.ch7.

[30] Samuel Kwong and Winton Yee. “Natural Language Generation of The Office Using a Neural Machine Translation Context”. en. In: ().

[31] O-Joun Lee, Nayoung Jo, and Jason Jung. “Measuring Character-based Story Similarity by Analyzing Movie Scripts”. In: Mar. 2018.

[32] S. Lee, H. Yu, and Y. Cheong. “Analyzing Movie Scripts as Unstructured Text”. In: *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*. 2017, pp. 249–254. DOI: 10.1109/BigDataService.2017.43.

[33] Brad Lindblad. *schrute: The Entire Transcript from the Office in Tidy Format*. R package version 0.2.2. 2020. URL: https://bradlindblad.github.io/schrute/.

[34] Andrew L. Maas et al. “Learning Word Vectors for Sentiment Analysis”. In: *ACL*. 2011.

[35] Sanidhya Mangal, Poorva Joshi, and Rahul Modak. “LSTM vs. GRU vs. Bidirectional RNN for script generation”. In: *CoRR* abs/1908.04332 (2019). arXiv: 1908.04332. URL: http://arxiv.org/abs/1908.04332.

[36] Stephen Merity, N. Keskar, and R. Socher. “An Analysis of Neural Language Modeling at Multiple Scales”. In: *ArXiv* abs/1803.08240 (2018).

[37] Fionn Murtagh, Adam Ganz, and Stewart McKie. “The structure of narrative: The case of film scripts”. In: *Pattern Recognition* 42.2 (2009). Learning Semantics from Multimedia Content, pp. 302–312. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2008.05.026. URL: http://www.sciencedirect.com/science/article/pii/S0031320308002100.

[38] Fionn Murtagh and Pierre Legendre. “Ward’s Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward’s Criterion?” In: *Journal of Classification* 31 (Oct. 2014), pp. 274–295. DOI: 10.1007/s00357-014-9161-z.

[39] Louis R. Nemzer and Florence Neymotin. “How words matter: machine learning & movie success”. In: *Applied Economics Letters* 27.15 (2020), pp. 1272–1276. DOI: 10.1080/13504851.2019.1676868. eprint: https://doi.org/10.1080/13504851.2019.1676868. URL: https://doi.org/10.1080/13504851.2019.1676868.

[40] Dong Nguyen, Noah A. Smith, and C. Rosé. “Author Age Prediction from Text using Linear Regression”. In: *LaTeCH@ACL*. 2011.

[41] Fred Popowich. "Using text mining and natural language processing for health care claims processing". In: *SIGKDD Explor.* 7 (2005), pp. 59–66.

[42] "Predicting the helpfulness of online reviews using a scripts-enriched text regression model". In: ().

[43] A. Radford et al. "Language Models are Unsupervised Multitask Learners". In: 2019.

[44] Tyler W. Rinker. *sentimentr: Calculate Text Polarity Sentiment*. version 2.7.1. Buffalo, New York, 2019. URL: http://github.com/trinker/sentimentr.

[45] Alexandra Schofield and David Mimno. "Comparing Apples to Apple: The Effects of Stemmers on Topic Models". In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 287–300.

[46] Dharak Shah and Saheb Motiani. "Movie Classification Using k-Means and Hierarchical Clustering An analysis of clustering algorithms on movie scripts". In: 2013.

[47] Julia Silge and David Robinson. *Text Mining with R: A Tidy Approach*. 1st. O'Reilly Media, Inc., 2017. ISBN: 1491981652.

[48] Ilya Sutskever, James Martens, and Geoffrey Hinton. "Generating Text with Recurrent Neural Networks". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, 2011, pp. 1017–1024. ISBN: 9781450306195.

[49] *Text Mining: Creating Tidy Text*. URL: https://uc-r.github.io/tidy_text.

[50] Abinash Tripathy, Ankit Agrawal, and Santanu Kumar Rath. "Classification of sentiment reviews using n-gram machine learning approach". In: *Expert Systems with Applications* 57 (2016), pp. 117–126. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2016.03.028. URL: http://www.sciencedirect.com/science/article/pii/S095741741630118X.

[51] *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[52] Marco Del Vecchio et al. "Improving productivity in Hollywood with data science: Using emotional arcs of movies to drive product and service innovation in entertainment industries". In: *Journal of the Operational Research Society* 0.0 (2020), pp. 1–28. DOI: 10.1080/01605682.2019.1705194. eprint: https://doi.org/10.1080/01605682.2019.1705194. URL: https://doi.org/10.1080/01605682.2019.1705194.

[53] Sholom M. Weiss. *Text mining : predictive methods for analyzing unstructured information*. English. Springer New York, 2005, xii, 237 p. : ISBN: 0387954333.

[54] Hongliang Yu, S. Zhang, and Louis-Philippe Morency. "Unsupervised Text Recap Extraction for TV Series". In: *EMNLP*. 2016.

[55] L. Zhang et al. "Combining lexicon-based and learning-based methods for twitter sentiment analysis". In: 2011.

7 APPENDIX

```
# Library ------
library(schrute)
library(tidyverse)
library(tidytext)
library(textrecipes)
library(textmineR)
library(textdata) # for data dictionary for sentiment analysis
library(textfeatures)
library(tidylo)
library(broom)
library(tidymodels)
library(gridExtra)
library(ggalt)
library(tm)
library(quanteda) # for nlp
library(tokenizers)
library(SnowballC)
library(spacyr)
library(stopwords)
library(wordcloud)
library(ggwordcloud)
library(quanteda.textmodels)
library(caret)
library(glmnet)
library(gt)
library(hardhat)
library(vip)
library(sentimentr)
library(ggraph)
library(igraph)
library(widyr)
library(topicmodels)
library(tictoc)
library(future)
library(keras)
conflicted::conflict_prefer("filter", "dplyr")
conflicted::conflict_prefer("vi", "vip")
conflicted::conflict_prefer("neighbors", "dials")
source('theme.R')

# EDA ---------------
data <- theoffice
glimpse(data) # 55,130 rows 12 col
# remove air_date, index, and text with direction
data <- data %>% select(-air_date, -index, -text_w_direction)
my_stopwords <- c("yeah", "hey", "uh", "um", "guy", "time", "go", "michael")
my_stopwords <- c(my_stopwords, "ah", "hmm", "huh", "ya", "'s", "ay", "ha", "je", "mo", "yy", letters)
my_stopwords <- c(my_stopwords, "ok", "well", "oh", "okay", "become", "every")

# * Data cleaning--------
# remove the parts 1&2 in episode_name
```

```r
data <- data %>% mutate(episode_name = str_squish(str_replace_all(episode_name, "\\(Parts 1&2\\)", "")))
# fix spelling mistakes
correct_spelling <- function(data, column, from, to) {
  ind <- which(data[,column] == from)
  data[ind, column] <- to
  return(data)
}

fr <- c("Charles McDougal", "Claire Scanlon", "Greg Daneils", "Ken Wittingham", "Paul Lieerstein")
to <- c("Charles McDougall", "Claire Scanlong", "Greg Daniels", "Ken Whittingham", "Paul Lieberstein")
for(i in seq_along(fr)) {
  data <- correct_spelling(data, "director", fr[i], to[i])
}
fr <- c("Michae", "MIchael", "Michal", "Micheal", "Michel", "Video Michael", "Warehouse Michael", "Young
for(i in seq_along(fr)) {
  data <- correct_spelling(data, "character", fr[i], "Michael")
}
# remove double quotations
data$character <- str_replace_all(data$character, '"', "")
data$character <- str_replace_all(data$character, " \\[on phone\\]", "")
# fix characters names
fr <- c("Deangelo", "David", "Packer", "Todd", "Daryl", "Ryan Howard", "Robert California", "Carroll",
to <- c("DeAngelo", "David Wallace", "Todd Packer", "Todd Packer", "Darryl", "Ryan", "Robert", "Carol",
for(i in seq_along(fr)) {
  data <- correct_spelling(data, "character", fr[i], to[i])
}
data <- data %>% mutate(director = str_squish(director), character = str_squish(character))


# * Ratings ------------

ratings <- data %>% select(season, episode, episode_name, director, writer, imdb_rating, total_votes) %
# hist(ratings$imdb_rating)
ratings %>% ggplot(aes(imdb_rating)) + geom_histogram(breaks = seq(6.5, 10, .5), fill = '#64A6B0', colo
  labs(title = 'Histogram of IMDB Ratings')

# Rating by Season
ratings %>% group_by(season) %>%
  summarise(no_of_ep = n(), avg_rating = mean(imdb_rating))
ratings %>%
  ggplot(aes(season, imdb_rating, fill = as.factor(season))) +
  geom_boxplot(show.legend = FALSE) +
  annotate("text", x = 7.9, y = 9, label = "Michael leaves") +
  annotate(
    geom = "curve", x = 8, y = 8.9, xend = 7.5, yend = 8.5,
    curvature = -.3, arrow = arrow(length = unit(2, "mm"))
  ) +
  scale_fill_manual(values = office_pal(length(unique(ratings$season)))) +
  scale_x_continuous(breaks = 1:9) +
  labs(x = 'Season', y = 'Rating', title = 'IMDB Ratings by Season')

# Rating by Episode
ratings %>% ggplot(aes(episode, imdb_rating, fill = as.factor(episode))) +
```

```r
  geom_boxplot(show.legend = FALSE) +
  scale_fill_manual(values = office_pal(length(unique(ratings$episode)))) +
  labs(x = "Episode Number", y = "Rating", title = "IMDB Ratings by Episode Number")

# Total Votes by Season
data %>% distinct(season, episode_name, total_votes, imdb_rating) %>%
  group_by(season) %>% summarise(`avg_votes` = round(mean(total_votes)))

# * Directors-----
ratings <- ratings %>% mutate(episode_number = row_number())
director <- ratings %>%
  select(season, episode, episode_name, director, imdb_rating, total_votes,
         episode_number) %>%
  distinct() %>%
  separate_rows(director, sep = ';') %>%
  add_count(director) %>%
  arrange(desc(n))

director %>% group_by(director, n) %>% summarise(`Average Rating` = round(mean(imdb_rating), 2)) %>%
  ungroup() %>% arrange(desc(n)) %>% slice(1:10)

director %>% group_by(director, n) %>% summarise(`Average Rating` = round(mean(imdb_rating), 2)) %>%
  ungroup() %>% arrange(desc(`Average Rating`)) %>% slice(1:10)

# TODO: FIX DIRECTOR_RATING WITH THE REST
director_rating <- director %>% group_by(director, n) %>% summarise(avg_rating = mean(imdb_rating))

director_rating %>% arrange(desc(avg_rating)) %>%
  mutate(director = fct_inorder(director), n2 = ifelse(n>5, n, NA)) %>%
  ggplot(aes(reorder(director, avg_rating), avg_rating)) +
  geom_segment(aes(yend = 7.0, xend = director), size = 0.2) +
  geom_point(aes(size = n, color = n > 5), show.legend = FALSE) +
  geom_text(aes(label = n2), size = 3, color = 'white') +
  scale_color_manual(values = c("gray40", beet)) +
  coord_flip() +
  labs(x = NULL, y = 'Average Rating', title = "Average IMDB Rating by Directors",
       subtitle = 'size represents number of episodes directed, \n color represents > 5 episodes directe
  theme(panel.grid.major.y = element_blank())



# * Writers------
writers <- ratings %>% select(season, episode, episode_name, writer, imdb_rating, total_votes, episode_r
  distinct() %>%
  separate_rows(writer, sep = ';') %>% add_count(writer) %>% arrange(desc(n))

# TODO: FIX WRITER RATINGS WITH THE REST
writer_rating <- writers %>% group_by(writer, n) %>% summarise(avg_rating = mean(imdb_rating))

writers %>% group_by(writer, n) %>%
  summarise(`Average Rating` = round(mean(imdb_rating),2)) %>%
  ungroup() %>% arrange(desc(n)) %>% slice(1:10) %>% rename('No. Episodes' = n)
```

```r
writers %>% group_by(writer, n) %>%
  summarise(`Average Rating` = round(mean(imdb_rating), 2)) %>%
  ungroup() %>% arrange(desc(`Average Rating`)) %>% slice(1:10) %>% rename('No. Episodes' = n)


writer_rating %>% arrange(desc(avg_rating)) %>%
  mutate(writer = fct_inorder(writer), n2 = ifelse(n>9, n, NA)) %>%
  ggplot(aes(reorder(writer, avg_rating), avg_rating)) +
  geom_segment(aes(yend = 6.5, xend = writer), size = 0.2) +
  geom_point(aes(size = n, color = n > 9), show.legend = FALSE) +
  geom_text(aes(label = n2), size = 3, color = 'white') +
  scale_color_manual(values = c("gray40", ofc)) +
  coord_flip() +
  labs(x = NULL, y = 'Average Rating', title = "Average IMDB Rating by Writers", subtitle = 'size repres
  theme(panel.grid.major.y = element_blank())


# * Characters------
length(unique(data$character))

characters <- data %>% count(character, sort = TRUE) # reduced to 715
# characters with more than 100 lines
main_char <- characters %>% filter(n > 100) %>%
  filter(!character %in% c("Pete", "Clark")) %>%
  pull("character")
main_char <- data %>% filter(character %in% main_char)
# choose characters that came out in more than 2 season
morethan2season <- main_char %>% select(season, character) %>% distinct() %>%
  count(character) %>% filter(n > 2) %>% pull(character)
main_char <- main_char %>% filter(character %in% morethan2season)
# total lines per episode per season
total_lines <- data %>% count(season, episode)
# number of lines per season
char_lines <- main_char %>% count(season, episode, character, name = "char_lines")
char_lines %>% left_join(total_lines) %>% mutate(prop = char_lines/n) %>% group_by(season, character) %
  summarise(avg_prop = mean(prop)) %>%
  ggplot(aes(season, avg_prop)) + geom_area(na.rm = TRUE, fill = ofc, alpha = 0.8, color = darkofc)  +
  geom_point(size = 0.5) +
  facet_rep_wrap(~reorder(character, -avg_prop), repeat.tick.labels = TRUE, ncol = 7) +
  scale_x_continuous(breaks = 1:9) +
  scale_y_continuous(labels = scales::percent) +
  labs(y = '', x = 'Season',
       title = 'Average Percentage of Lines per Season for each Main Character',
       subtitle = 'Characters appearing in more than two seasons and with more than 100 lines in total'
  theme(panel.grid.minor.x = element_blank())

# percentage of episodes each character showed up in
total_ep <- length(unique(data$episode_name))
main_char %>% distinct(episode_name, character) %>% count(character) %>% mutate(prop_appear = n/total_ep
  mutate(character = fct_reorder(character, prop_appear)) %>%
  ggplot(aes(prop_appear, character)) + geom_segment(aes(xend = 0, yend = character), size = 0.2) + geom
  labs(x = 'Percentage of Appearance', y = NULL, title = 'Character Appearances in The Office') +
  theme(panel.grid.major.y = element_blank())
```

```r
# * Text EDA------
# (did not use these two plots in the paper)
data %>% group_by(episode_name) %>% summarise(text = paste0(text, collapse = " ")) %>%
  mutate(char_count = nchar(text)) %>%
  ggplot(aes(char_count)) + geom_histogram(bins = 20, alpha = 0.8) +
  scale_x_log10() +
  labs(x = 'Number of characters', title = "Distribution of Character Counts")
data %>% group_by(episode_name) %>% summarise(text = paste0(text, collapse = " ")) %>%
  unnest_tokens(word, text) %>%
  count(episode_name) %>%
  ggplot(aes(n)) + geom_histogram(bins = 25) +
  scale_x_log10() +
  labs(x = "Number of Words per Episode", y = "Number of Episodes", title = "Distribution of Word Count
#-------


# Character TF-IDF: which words are specific to characters------
# lemmatization
spacy_initialize(entity = FALSE)
char_parse <- main_char %>% mutate(doc_id = paste0("doc", row_number()),
                                   text = str_replace_all(text, "-", " "),
                                   text = str_replace_all(text, '"', " ")) %>%
  select(doc_id, text) %>%
  spacy_parse() %>%
  filter(!pos %in% c("SCONJ", "PART", "PRON", "DET", "NUM", "SPACE","PUNCT")) %>%
  filter(str_detect(lemma, "'d", negate = TRUE)) %>%
  mutate(lemma = str_to_lower(lemma)) %>%
  anti_join(get_stopwords(), by = c("lemma" = "word"))

char_lemma <- char_parse %>% select(doc_id, lemma) %>% rename(word = lemma)
char_text_tfidf <- main_char %>% mutate(doc_id = paste0("doc", row_number())) %>% left_join(char_lemma)
# remove periods and empty strings
char_text_tfidf <- char_text_tfidf %>% mutate(word = str_replace_all(word, "\\.", "")) %>% filter(word
tokeep <- main_char %>% count(character) %>% mutate(perc_line = n/nrow(main_char)) %>%
  filter(n > 600) %>% pull(character)
char_text_tfidf %>%
  filter(character %in% unique(tokeep)) %>%
  mutate(character = as.factor(character)) %>%
  count(character, word, sort = TRUE) %>%
  bind_tf_idf(word, character, n) %>%
  group_by(character) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(character = as.factor(character),
         word = reorder_within(word, tf_idf, character)) %>%
  ggplot(aes(word, tf_idf, fill = character)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~character, scales = 'free', ncol = 4) +
  scale_x_reordered() +
  coord_flip() +
  scale_fill_manual(values = office_pal(length(unique(tokeep)))) +
  theme(strip.text = element_text(size = 9), axis.text.x = element_text(size = rel(0.6), face = 'bold')
```

```
        axis.text.y = element_text(size = 8)) +
  labs(x = '', y = 'tf-idf', title = "Words specific to characters based on tf-idf")


# correlation between number of lines of main characters and imdbrating
# calculate percent of word use across all novels
office_pct <- data %>%
  unnest_tokens(word, text, strip_punct = TRUE, strip_numeric = TRUE) %>%
  anti_join(stop_words) %>%
  count(word) %>%
  transmute(word, all_words = n / sum(n))
# calculate percent of word use within each season
frequency <- data %>%
  unnest_tokens(word, text, strip_punct = TRUE, strip_numeric = TRUE) %>%
  mutate(word = textstem::lemmatize_words(word)) %>%
  anti_join(stop_words) %>%
  filter(!word %in% my_stopwords) %>%
  filter(str_detect(word, "[:digit:]", negate = TRUE)) %>%
  count(season, word)
frequency <- frequency %>%
  group_by(season) %>%
  summarise(total_season = sum(n)) %>%
  ungroup() %>%
  left_join(frequency) %>%
  mutate(season_words = n/total_season) %>%
  left_join(office_pct) %>%
  arrange(desc(season_words))

ggplot(frequency, aes(x = season_words, y = all_words, color = abs(all_words - season_words))) +
  geom_abline(color = "gray40", lty = 2) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.5, height = 0.5) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1) +
  scale_x_log10(labels = scales::percent_format()) +
  scale_y_log10(labels = scales::percent_format()) +
  scale_color_gradient(limits = c(0, 0.01), low = "darkslategray4", high = "gray75") +
  facet_wrap(~ season, ncol = 3) +
  theme(legend.position="none") +
  labs(x = "Word frequency in season", y = "Word frequency in all season",
       title = "Word frequency across entire show vs within each season")



# Sentiment Analysis --------
core_char <- main_char %>% filter(character %in% tokeep, text != "") %>% select(season, episode, episod

season_char_text_df <- core_char %>% group_by(character, season) %>%
  summarise(text = paste0(text, collapse = " ")) %>% ungroup() %>%
  mutate(element_id = row_number())
season_char_sentence <- season_char_text_df %>% get_sentences(text)
season_char_text <- season_char_sentence %>%
  sentiment(polarity_dt = lexicon::hash_sentiment_sentiword) %>%
  sentiment_by(averaging.function = average_weighted_mixed_sentiment)
```

```r
char_text <- sentiment_by(season_char_text, by = season_char_text_df$character,
                          averaging.function = average_weighted_mixed_sentiment)

# plot(season_char_text)
season_char_text_df %>% select(character, season, element_id) %>%
  left_join(season_char_text %>% select(element_id, ave_sentiment)) %>%
  left_join(char_text, by = "character") %>%
  ggplot(aes(ave_sentiment.x, reorder(character, -ave_sentiment.y))) +
  geom_path(color = "#6376A8", size = 0.4) +
  geom_point(size = 2, shape = 21, fill = "#6376A8", alpha = 0.5) +
  geom_text(aes(label = ifelse(ave_sentiment.x < -0.25 | ave_sentiment.x > 0, season, "")), vjust = -1,
  geom_point(aes(ave_sentiment, character), data = char_text, size = 3, shape = 21, fill = "#F2AE54") +
  labs(x = "Average Sentiment Score", y = NULL, title = "Sentiment Polarity of each Character throughou

emotions <- core_char %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word") %>%
  filter(!grepl('[0-9]', word)) %>%
  left_join(get_sentiments("nrc"), by = "word") %>%
  filter(!(sentiment == "negative" | sentiment == "positive")) %>%
  group_by(character, sentiment) %>%
  summarize( freq = n()) %>%
  mutate(percent=round(freq/sum(freq)*100)) %>%
  select(-freq) %>%
  spread(sentiment, percent, fill=0) %>%
  ungroup()
## Normalize data
sd_scale <- function(x) {
  (x - mean(x))/sd(x)
}
emotions[,c(2:9)] <- scale(emotions[,c(2:9)])
emotions <- as.data.frame(emotions)
rownames(emotions) <- emotions[,1]
emotions3 <- emotions[,-1]
emotions3 <- as.matrix(emotions3)
## Using a heatmap and clustering to visualize and profile emotion terms expression data

gplots::heatmap.2(
  emotions3,
  dendrogram = "both",
  scale      = "none",
  trace      = "none",
  key        = TRUE,
  col    = colorRampPalette(c("#74CFEB", "#EFD922", "#AE484B")),
  cexCol = 0.85,
  cexRow = 0.8
)

core_char %>% unnest_tokens(word, text) %>%
  anti_join(stop_words) %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  cast_dfm(sentiment, word, n) %>%
```

```
  textplot_wordcloud(comparison = TRUE, max_words = 100, color = c("#E8696C", "#30613B"),
                     random_order = FALSE, font = "StaffMeetingPlain",
                     labelcolor = darkofc)

# Clustering -------
text_df <- main_char %>% group_by(character) %>% summarise(text = paste0(text, collapse = " ")) %>% ungr

dtm <- text_df %>%
  unnest_tokens(word, text, strip_numeric = TRUE) %>%
  mutate(word = SnowballC::wordStem(word)) %>%
  anti_join(stop_words) %>%
  count(doc_id, word) %>%
  cast_dtm(doc_id, word, n, weighting = tm::weightTfIdf)

# Remove sparse terms
dtm <- removeSparseTerms(dtm,sparse=0.90)
dtm_tx <- weightTfIdf(dtm)
mat_dtm <- proxy::as.matrix(dtm_tx)

# Term Normalization
normalise_dtm <- function(y) {y/apply(y, MARGIN=1,
                                      FUN = function(k) {sum(k^2)^.5})}
dtm_norm <- normalise_dtm(mat_dtm)
distance <- proxy::dist(dtm_norm, method="cosine")
hc <- hclust(distance, method="ward.D2")

library(factoextra)
fviz_dend(hc, cex = 1,
          k = 10,
          k_colors = office_pal(10),
          color_labels_by_k = TRUE,
          main = "Dendrogram of Main Characters",
          xlab = "Main Characters",  ylab = "Cosine similarity",
          sub = "",
          horiz = FALSE,
          rect = TRUE,
          labels_track_height = 0.1,
          ggtheme = theme_classic(base_family = "StaffMeetingPlain"))

# Regression------
# * Data preparation-------

# Feature engineering:
# season:
season_features <- data %>% distinct(episode_name, season) %>% transmute(episode_name, feature = paste(
# episode:
episode_features <- data %>% distinct(episode_name, episode) %>% transmute(episode_name, feature = paste
# director & writer:
director_writer_features <- data %>% distinct(episode_name, director, writer) %>%
  gather(type, value, director, writer) %>%
  separate_rows(value, sep = ";") %>%
  unite(feature, type, value, sep = ": ") %>%
  group_by(feature) %>%
```

```
  filter(n() > 1) %>%
  mutate(value = 1) %>%
  ungroup()

# main characters:
character_features <- main_char %>% distinct(episode_name, character, text) %>%
  count(episode_name, character) %>%
  transmute(episode_name, feature = character, value = n)

# each episode is considered as one document
# Pre-processing of Text Data before any feature engineering and modeling:
text_docs <- data %>% group_by(episode_name) %>% summarise(text = paste0(text, collapse = " ")) %>% ungr

# tokenize using spacy to lemmatize
spacy_initialize(entity = FALSE)
text_tokens <- text_docs %>% mutate(doc_id = paste0("doc", row_number()),
                                    text = str_replace_all(text, "-", " "),
                                    text = str_replace_all(text, '"', " "),
                                    text = str_replace_all(text, "/", " "),
                                    text = str_replace_all(text, "[^[:graph:]]", " ")) %>%
  select(doc_id, text) %>%
  spacy_parse()

pos_id <- text_tokens %>% distinct(pos)
text_tokens$`pos`[which(text_tokens$`pos` == "PART" & text_tokens$`lemma` == "s*x")] <- "VERB"
text_tokens <- text_tokens %>%
  mutate(lemma = str_to_lower(lemma),
         lemma = str_replace_all(lemma, "[\\.']", "")) %>%
  filter(!pos %in% c("PUNCT", "SPACE", "AUX", "CCONJ", "NUM", "SYM", "PART")) %>%
  filter(!lemma %in% c("-pron-")) %>%
  select(doc_id, lemma) %>%
  group_by(doc_id) %>%
  summarise(text = paste0(lemma, collapse = " ")) %>% ungroup()

text_docs <- text_docs %>% mutate(doc_id = paste0("doc", row_number())) %>% select(-text) %>%
  left_join(text_tokens) %>% select(-doc_id)

# add original text for count features:
original_text <- data %>% group_by(episode_name) %>% summarise(text = paste0(text, collapse = " ")) %>%

features <- bind_rows(director_writer_features, season_features, episode_features, character_features) %
  pivot_wider(names_from = feature, values_from = value, values_fill = list(value = 0)) %>%
  left_join(text_docs) %>%
  left_join(original_text) %>%
  left_join(data %>% distinct(season, episode_name, imdb_rating, total_votes))

# Data splitting 80/20
set.seed(1234)
office_split <- initial_split(features, prop = 4/5, strata = season)
office_train <- training(office_split)
office_test <- testing(office_split)

# cross validation:
```

```r
set.seed(123)
office_folds <- vfold_cv(office_train, strata = season)

# * Recipe-------
# Pre-processing specification:
office_recipe <- recipe(imdb_rating ~ ., data = office_train) %>%
  step_rm(season) %>%
  update_role(episode_name, new_role = "ID") %>%
  step_textfeature(original_text) %>%
  step_tokenize(text, token = "ngrams", options = list( #unigram + bigram
    n = 2, n_min = 1,
    stopwords = c(stopwords(source = "snowball"), my_stopwords)
  )) %>%
  step_tokenfilter(text, max_tokens = 1000, min_times = 5) %>%
  step_tfidf(text) %>%
  step_zv(all_predictors()) %>%
  step_corr(all_predictors()) %>%
  step_normalize(all_predictors())

# * LASSO ------
# very sparse data
library(hardhat)
sparse_bp <- default_recipe_blueprint(composition = "dgCMatrix")

# Model specification:
tune_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

# Combine preprocessing and model spec into tunable workflow:
tune_wf <- workflow() %>%
  add_recipe(office_recipe, blueprint = sparse_bp) %>%
  add_model(tune_spec)
tune_wf

doParallel::registerDoParallel()
set.seed(2020)
# tune hyperparameter using cross val
office_rs <- tune_grid(tune_wf,
                       office_folds,
                       grid = 20,
                       metrics = metric_set(rmse, mae, mape)
)

office_rs %>% collect_metrics() %>%
  ggplot(aes(penalty, mean, color = .metric)) +
  geom_errorbar(aes(
    ymin = mean - std_err, ymax = mean + std_err), alpha = 0.5, width = 0.1) +
  geom_line(size = 1.5) + facet_wrap(~.metric, scales = "free", nrow = 3) +
  scale_x_log10() +
  theme(legend.position = "none") +
  labs(title = "Lasso model performance across regularization penalties",
       x = "penalty (log)") +
```

```
  scale_color_manual(values = office_pal(3))

# best model:
lowest_rmse <- office_rs %>% select_best("rmse")
office_rs %>% show_best("rmse")
train_valid <- office_rs %>% show_best("rmse", n = 1) %>%
  select(mean, std_err) %>% mutate(model = "LASSO")

# Finalize workflow:
final_wf <- finalize_workflow(
  tune_wf, lowest_rmse
)
final_wf

# coefficients
lasso_coef <- final_wf %>% fit(office_train) %>% pull_workflow_fit() %>% tidy()
lasso_coef %>% filter(estimate != 0) # went from 1,121 variables to 68

# Fit our full training data and evaluate on test data
final_lasso <- last_fit(final_wf, office_split)
collect_metrics(final_lasso)

# Important variables:
office_imp <- pull_workflow_fit(final_lasso$.workflow[[1]]) %>%
  vi(lambda = lowest_rmse$penalty) %>%
  mutate(
    Variable = str_remove_all(Variable, "tfidf_text_"),
    Variable = str_remove_all(Variable, "textfeature_original_text_"),
    Variable = fct_reorder(Variable, Importance)
  )

office_imp %>%
  group_by(Sign) %>%
  top_n(10, abs(Importance)) %>%
  ungroup() %>%
  ggplot(aes(x = Importance, y = Variable, fill = Sign)) +
  geom_col(show.legend = FALSE) +
  scale_fill_manual(values = c(beet, ofc)) +
  labs(y = NULL, title = "Variable Importance by LASSO")


office_imp %>%
  filter(str_detect(Variable, "episode|writer|director|season|^[A-Z]|total_votes")) %>%
  group_by(Sign) %>%
  top_n(7, abs(Importance)) %>%
  ungroup() %>%
  ggplot(aes(x = Importance, y = Variable, fill = Sign)) +
  geom_col(show.legend = FALSE) +
  scale_fill_manual(values = c(beet, ofc)) +
  labs(y = NULL, title = "Variable Importance by LASSO", subtitle = "non-text features")


final_lasso %>%
```

```r
  collect_predictions() %>%
  ggplot(aes(imdb_rating, .pred)) +
  geom_point(alpha = 0.8) +
  labs(x = 'Observed IMDB Rating', y = 'Predicted IMDB Rating',
       title = "Predicted vs. observed IMDB ratings for the test data set of \n The Office transcripts u
  theme(plot.title = element_text(size = 12)) +
  coord_obs_pred()

# * SVR-----

# Model specification:
tune_spec_svm <- svm_poly(cost = tune(), degree = tune(), scale = tune(),
                          margin = tune()) %>%
  set_mode("regression") %>%
  set_engine("kernlab")
tune_spec_svm

# Combine preprocessing and model spec into tunable workflow:
tune_svm_wf <- workflow() %>%
  add_recipe(office_recipe) %>%
  add_model(tune_spec_svm)
tune_svm_wf

doParallel::registerDoParallel() # 12:54-1:09
set.seed(2020)
office_rs_svm <- tune_grid(tune_svm_wf,
                           office_folds,
                           grid = 10,
                           metrics = metric_set(rmse, mae, mape)
)

office_rs_svm %>% collect_metrics() %>%
  mutate(cost = log2(cost), scale_factor = log10(scale_factor)) %>%
  rename(`Cost (log-2)` = cost, `Polynomial Degree` = degree,
         `Scale Factor (log-10)` = scale_factor, `Insensitivity Margin` = margin) %>%
  pivot_longer(cols = 1:4) %>%
  ggplot(aes(value, mean, color = name)) +
  geom_point(show.legend = FALSE) +
  geom_errorbar(aes(ymin = mean - std_err, ymax = mean + std_err),
                alpha = 0.5, width = 0, show.legend = FALSE) +
  facet_grid(.metric ~ name, scales = "free") +
  theme(panel.border = element_rect(fill = NA)) +
  labs(
    title = "Support vector regression performance across different hyperparameters",
    x = NULL
  ) +
  scale_color_manual(values = office_pal(4))


# best model:
lowest_rmse_svm <- office_rs_svm %>% select_best("rmse")
office_rs_svm %>% show_best("rmse", n = 1)
train_valid <- office_rs_svm %>% show_best("rmse", n = 1) %>% select(mean, std_err) %>%
```

```
  mutate(model = "SVR") %>% bind_rows(train_valid)

# Finalize workflow:
final_svm_wf <- finalize_workflow(
  tune_svm_wf, lowest_rmse_svm
)
final_svm_wf

# Fit our full training data and evaluate on test data
final_svm <- last_fit(final_svm_wf, office_split)
collect_metrics(final_svm)

# * XBGoost------
# Model specification:
office_recipe_xgb <- recipe(imdb_rating ~ ., data = office_train) %>%
  step_rm(season) %>%
  update_role(episode_name, new_role = "ID") %>%
  step_textfeature(original_text) %>%
  step_tokenize(text, token = "ngrams", options = list( #unigram + bigram
    n = 2, n_min = 1,
    stopwords = c(stopwords(source = "snowball"), my_stopwords)
  )) %>%
  step_tokenfilter(text, max_tokens = 1000, min_times = 5) %>%
  step_tfidf(text) %>%
  step_zv(all_predictors()) %>%
  step_corr(all_predictors())


tune_spec_xgb <- boost_tree(trees = tune(), mtry = tune(), tree_depth = tune(),
                            learn_rate = tune()) %>%
  set_mode("regression") %>%
  set_engine("xgboost")

# Combine preprocessing and model spec into tunable workflow:
tune_xgb_wf <- workflow() %>%
  add_recipe(office_recipe_xgb) %>%
  add_model(tune_spec_xgb)
tune_xgb_wf

# cross validation:
doParallel::registerDoParallel() # 3:55
set.seed(2020)
office_rs_xgb <- tune_grid(tune_xgb_wf,
                           office_folds,
                           grid = 20,
                           metrics = metric_set(rmse, mae, mape)
)


office_rs_xgb %>% collect_metrics() %>%
  mutate(learn_rate = log10(learn_rate)) %>%
  rename(`# Randomly Selected Predictors` = mtry,
         `# Trees` = trees,
```

```
         `Learning Rate (log-10)` = learn_rate,
         `Tree Depth` = tree_depth) %>%
  pivot_longer(cols = 1:4) %>%
  ggplot(aes(value, mean, color = .metric)) +
  geom_point(show.legend = FALSE) +
  geom_errorbar(aes(ymin = mean - std_err, ymax = mean + std_err),
                alpha = 0.5, show.legend = FALSE) +
  facet_grid(.metric ~ name, scales = "free") +
  theme(panel.border = element_rect(fill = NA)) +
  labs(title = "XGBoost Performance across Different Hyperparameters",
       x = NULL, y = NULL) +
  scale_color_manual(values = office_pal(3))
# errorbars are very small

# best model:
lowest_rmse_xgb <- office_rs_xgb %>% select_best("rmse")
office_rs_xgb %>% show_best("rmse")
train_valid <- office_rs_xgb %>% show_best("rmse", n = 1) %>% select(mean, std_err) %>%
  mutate(model = "XGBoost") %>% bind_rows(train_valid)

# Finalize workflow:
final_xgb_wf <- finalize_workflow(
  tune_xgb_wf, lowest_rmse_xgb
)
final_xgb_wf

xgb_vip <- final_xgb_wf %>%
  fit(data = office_train) %>%
  pull_workflow_fit() %>%
  vi()
xgb_vip %>%
  mutate(Variable = str_remove_all(Variable, "tfidf_text_"),
         Variable = str_remove_all(Variable, "textfeature_original_text_")) %>%
  vip(aesthetics = list(fill = ofc, alpha = 0.8, color = ofc)) +
  labs(title = "Top 10 Important Variables by XGBoost",
       subtitle = "Lowercase words are spoken text")

xgb_vip %>%
  filter(str_detect(Variable, "tfidf_text_", negate = TRUE),
         str_detect(Variable, "textfeature_original_text_", negate = TRUE)) %>%
  vip(aesthetics = list(fill = ofc, alpha = 0.8, color = ofc)) +
  labs(title = "Top 10 Important Variables by XGBoost",
       subtitle = "Excluding text variables")

# Fit our full training data and evaluate on test data
final_xgb <- last_fit(final_xgb_wf, office_split)
collect_metrics(final_xgb)



final_xgb %>%
  collect_predictions() %>%
  ggplot(aes(imdb_rating, .pred)) +
```

```
  geom_abline(lty = 2, color = "gray80", size = 1.5) +
  geom_point(alpha = 0.8) +
  labs(x = 'Observed IMDB Rating', y = 'Predicted IMDB Rating', title = "Predicted vs. observed IMDB ra
  theme(plot.title = element_text(size = 12))

# * KNN------
# Model specification:
tune_spec_knn <- nearest_neighbor(neighbors = tune(), dist_power = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")

# Combine preprocessing and model spec into tunable workflow:
tune_knn_wf <- workflow() %>%
  add_recipe(office_recipe) %>%
  add_model(tune_spec_knn)
tune_knn_wf

knn_param <- tune_knn_wf %>%
  parameters() %>%
  Matrix::update(
    neighbors = neighbors(c(3, 129))
  )
ctrl <- control_bayes(verbose = TRUE)

# cross validation:
doParallel::registerDoParallel()
set.seed(2020)
# bayesian optimization
office_rs_knn <- tune_bayes(tune_knn_wf,
                            office_folds,
                            initial = 5,
                            iter = 30,
                            param_info = knn_param,
                            control = ctrl,
                            metrics = metric_set(rmse, mae, mape)
)

office_rs_knn %>% collect_metrics() %>%
  rename(`# Nearest Neighbors` = neighbors, `Minkowski Distance Order` = dist_power) %>%
  pivot_longer(cols = c(1, 2)) %>%
  ggplot(aes(value, mean, color = .metric)) + geom_point(show.legend = FALSE) +
  geom_errorbar(aes(ymin = mean - std_err, ymax = mean + std_err), width = 0, show.legend = FALSE) +
  facet_grid(.metric~name, scales = "free") +
  theme(panel.border = element_rect(fill = NA)) +
  labs(x = NULL, y = NULL, title = "kNN Performance across Different Hyperparameters") +
  scale_color_manual(values = office_pal(3))

office_rs_knn %>% show_best("rmse")


# best model:
lowest_rmse_knn <- office_rs_knn %>% select_best("rmse")
train_valid <- office_rs_knn %>% show_best("rmse", n = 1) %>% select(mean, std_err) %>%
```

```r
  mutate(model = "kNN") %>% bind_rows(train_valid)

# Finalize workflow:
final_knn_wf <- finalize_workflow(
  tune_knn_wf, lowest_rmse_knn
)

# Fit our full training data and evaluate on test data
final_knn <- last_fit(final_knn_wf, office_split)
collect_metrics(final_knn)


final_results <- list(final_lasso, final_svm, final_xgb, final_knn) %>%
  map(collect_metrics) %>% bind_rows() %>%
  mutate(model = rep(c("LASSO", "SVR", "XGBoost", "kNN"), each = 2)) %>%
  select(model, .metric, .estimate) %>%
  pivot_wider(names_from = .metric, values_from = .estimate)

# * Neural net---------
# Recipe for NN
simple_recipe <- function(dataset){
  recipe(imdb_rating ~ ., data = dataset) %>%
    step_rm(season) %>%
    step_rm(episode_name) %>%
    step_textfeature(original_text) %>%
    step_tokenize(text, token = "ngrams", options = list( #unigram + bigram
      n = 2, n_min = 1,
      stopwords = c(stopwords(source = "snowball"), my_stopwords)
    )) %>%
    step_tokenfilter(text, max_tokens = 1000, min_times = 5) %>%
    step_tfidf(text) %>%
    step_zv(all_predictors()) %>%
    step_corr(all_predictors()) %>%
    step_range(all_predictors())
}

library(neuralnet)
# prep and bake training data and testing data
office_nn_train <- prep(simple_recipe(office_train), training = office_train)
office_nn_juice <- juice(office_nn_train) %>% janitor::clean_names()
feat = names(office_nn_juice)
form = as.formula(paste("imdb_rating~", paste(feat[!feat %in% "imdb_rating"], collapse = "+"), collapse
office_nn_test <- bake(office_nn_train, new_data = office_test) %>%
  janitor::clean_names()

# cross validation
nn_cv <- function(hidden_units, split, id){
  analysis_set <- analysis(split)
  analysis_prep <- prep(simple_recipe(analysis_set), training = analysis_set)
  analysis_processed <- juice(analysis_prep) %>%
    janitor::clean_names()
  feat = names(analysis_processed)
  form = as.formula(paste("imdb_rating~", paste(feat[!feat %in% "imdb_rating"],
```

```
                                                  collapse = "+"), collapse = "+"))
  model <- neuralnet(form, analysis_processed, hidden = hidden_units,
                     linear.output = TRUE,
                     err.fct = "sse")
  assessment_set <- assessment(split)
  assessment_processed <- bake(analysis_prep, new_data = assessment_set) %>%
    janitor::clean_names()
  tibble::tibble("id" = id,
                 "truth" = assessment_processed$imdb_rating,
                 "prediction" = unlist(predict(model, newdata = assessment_processed)))
}

# cross validation given hidden units
results_example <- map2_df(.x = office_folds$splits,
                           .y = office_folds$id,
                           ~nn_cv(c(100, 50, 50, 25), split = .x, id = .y))

train_valid <- results_example %>%
  group_by(id) %>%
  summarise(rmse = sqrt(mean((truth-prediction)^2))) %>% ungroup() %>%
  summarise(mean = mean(rmse), std_err = sd(rmse)) %>%
  mutate(model = "Neural Net") %>%
  bind_rows(train_valid)

# evaluate on test data
office_nn_mod <- neuralnet(form, office_nn_juice, c(100,50,50,25),
                           linear.output = TRUE,
                           err.fct = "sse")
final_nn_pred <- tibble::tibble("truth" = office_nn_test$imdb_rating,
                                "prediction" = unlist(predict(office_nn_mod,
                                                              newdata = office_nn_test)))

final_nn_pred %>%
  mutate(errors = (truth - prediction)^2) %>%
  ggplot(aes(truth, prediction)) + geom_point() +
  geom_abline(lty = 2) +
  labs(title = "Predicted vs. Actual")

final_nn_pred %>%
  mutate(errors = (truth - prediction)) %>%
  ggplot(aes(prediction, errors)) + geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Fitted vs. Residuals")


final_nn_pred %>%
  summarise(rmse = sqrt(mean((truth - prediction)^2))) %>%
  mutate(model = "Neural Net") %>%
  select(model, everything()) %>%
  bind_rows(final_results)

# * Training/Validation--------
train_valid
```

```r
train_valid %>%
  arrange(mean) %>%
  mutate(model = as_factor(model)) %>%
  ggplot(aes(model, mean, color = model)) +
  geom_point(show.legend = FALSE, size = 3) +
  geom_errorbar(aes(ymin = mean - std_err, ymax = mean + std_err), size = 1,
                width = 0.5,
                show.legend = FALSE) +
  labs(x = NULL, y = "rmse", title = "Model performance on training data") +
  scale_color_manual(values = office_pal(5))


final_results
```