Who Said What & Topic Modeling Stranger Things

Authors: Jacqueline Skunda, Malini Varadarajan, Tamara Qawasmeh
Coach: Professor Eric Gilbert, Ph.D
Fall 2022 - Team 36
UNIVERSITY OF MICHIGAN

Introduction

Stranger Things is a Netflix series with an extremely popular social media following. And the series has a huge social media presence making it a very interesting topic. TV shows such as these have teleplays which are scripts where the different characters enact the series. We wanted to use the Stranger things show and its datasets for various Supervised machine-learning approaches. Our goal was to build an algorithm that can identify which character in the series said what regarding the Stranger Things scripts (teleplays). We also wanted to explore the r/Stranger Things subreddit data to perform some unsupervised learning tasks such as topic modeling to find topics of interest within the forum. Note we have no stakeholders involved in this project as we are doing this out of our pure curiosity and fascination with the Netflix series and are fans ourselves.

Part A: Supervised Learning

Motivation

Supervised learning uses labeled datasets to train algorithms on how to classify data or make accurate predictions. We extracted the data from scripts (teleplays) of Stranger Things episodes that were rendered as pdf and converted them into dataframes. Where each row is a character and a dialog spoken by the said character. This type of labeled dataset is a great starting point for the model training. The model was trained with the dataset where each character speaks their dialogs, and the model learns the characteristics of how each character speaks. The trained model can essentially predict what type of dialog a character tends to speak. We went in knowing that extracting data from PDFs can be cumbersome, and it was since we had to read the row line by line and convert lines spoken by a character in an episode into a dictionary which was not easy to produce given how awkward the formatting of PDFs are. Ideally though, we wanted to use a novel dataset such as a script pdf to extract labeled data that we trained the model on, and really learn what challenges we would encounter and address them as we go.

Data Source

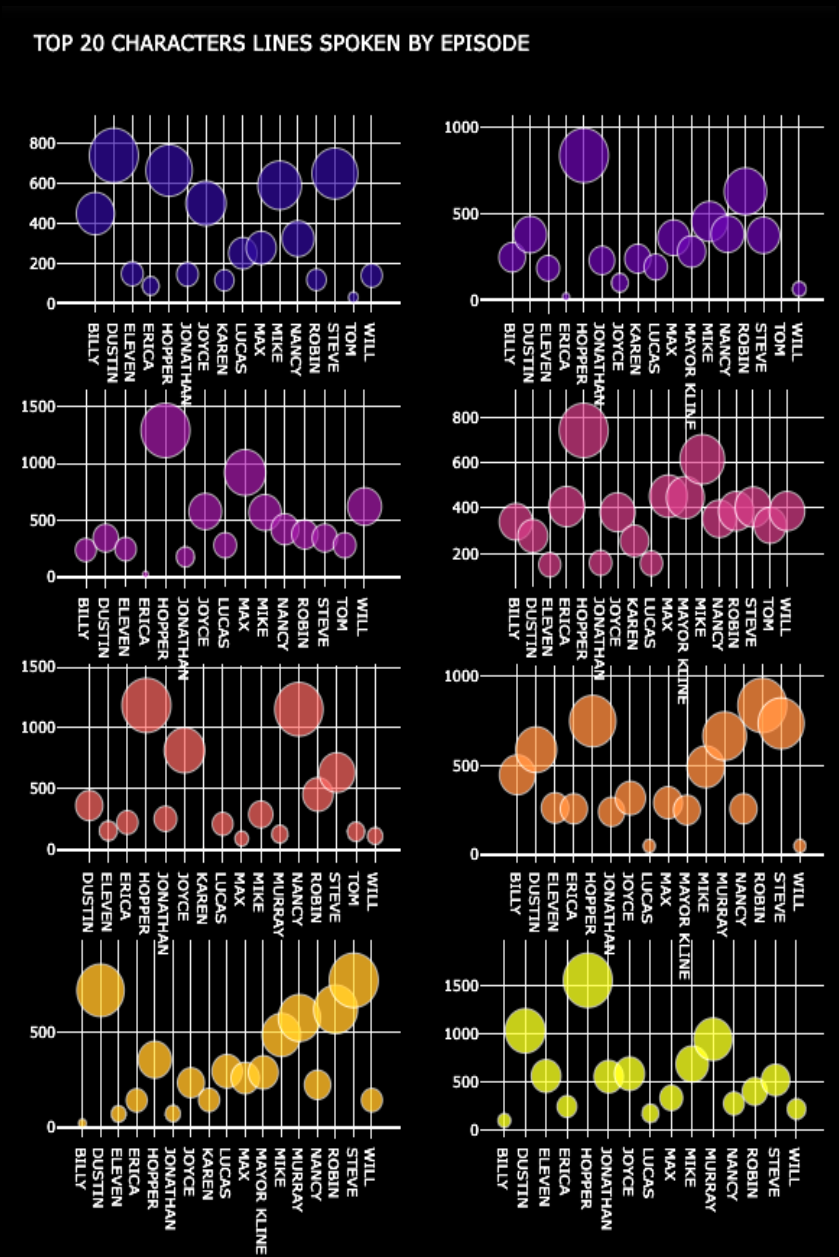
We are using publicly available PDF teleplays from the website 8flix (https://8flix.com). Scripts for available seasons (Full Season 3 and parts of seasons 1,2 and 4) were downloaded from the above website. We also used Wikipedia to get the main and recurring character names to only keep lines spoken by main or recurring characters so the classifier will have a more balanced representation of each character and enough dialog per character to learn from. The PDFs are in a form where the characters dialog and scene dialog are centered on the page and before any character speaks their name is placed in all uppercase letters. This identification was key for being able to iterate through the pdf and pull out dialog per character. Besides having the character name in their text spoken as a feature, having the season and episode number for the line spoken also proved to be helpful. To get our dataset created we had to do quite a bit of preprocessing with the PDFs to extract labeled character dialogs. Initially PDFMiner was used to extract all text from scripts we were able to download. We defined a function where the dictionary keys are the names of the characters. And the values are the lines spoken by the characters to read in and help parse the scripts for what we needed. For every line spoken in the series, we tag the character that speaks the line and we use this data to train the classifier properly. For example, when speaking at the said moment, a character could say one sentence or four sentences but regardless that is their said spoken line. Therefore, a character line refers to when they are speaking in the script but does not imply only one phrase or sentence being spoken, it could be one word or multiple sentences.

When we created the dictionary, as stated above, we used the name of each character as a key and added lines as the value for every time a character spoke, after going through and marking any uppercase token that occurred after 2 newlines with double colons via regex. When identifying the uppercase word surrounded by colons the words/lines spoken were added as a list for said character's key in the dictionary along with the season and episode number at the end of each pdf/document iteration and then converted to a dataframe and then combined into one large dataframe. Each list was exploded onto its own row to retrieve the intended character line spoken. While iterating through, we also had to fix misspelled character names along with identifying if the character was a main or recurring character and when they appeared to make sure our list of characters was robust. We used this during our classifier training process to help improve the accuracy of the learner.

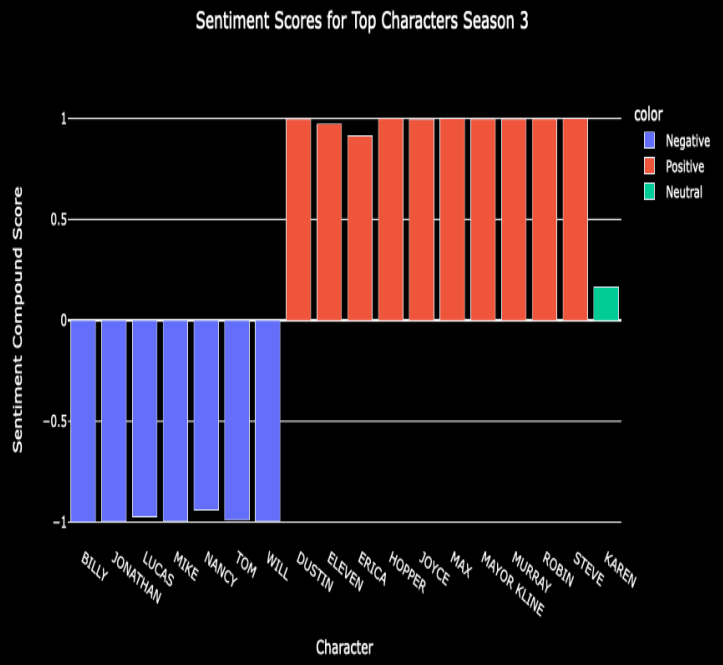
The function we created to do this initially returned 6,068 spoken lines of dialog for characters we identified via Wikipedia along with spelling correction/name variance adjustments. After cleaning down any oddly collected lines we arrived at an initial dataset with cleaned character lines of 5,451 for our 3 classification models. We then went in and add features such as gender, age, sentiment compound score, sentiment value (based on compound score), length of a line spoken (based on token count), total lines spoken, text difficulty based on line spoken, grade level based on line spoken along with a Top 20 tag (based on total lines spoken in the entire dataset). These various features were used to help train the models to improve their learning overall as well as used as filters to remove characters who only appeared to speak 2 times or less in the series or whose line of dialog spoken was 3 words or less.

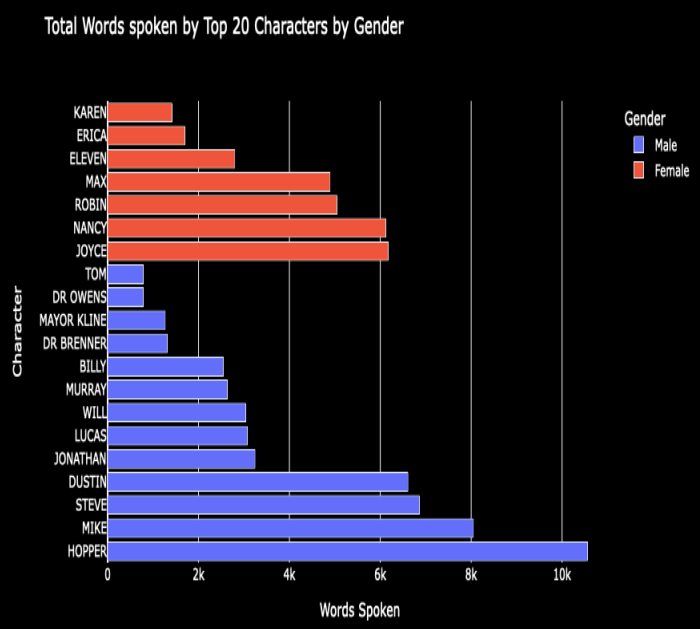
EDA

The initial analysis of the dataset was our way of learning about the type of data we were dealing with. Upon examining the data set after cleaning as much as possible we started to inspect the class labels, the Characters, and the number of words and number of lines (aka Dialog) spoken by the top characters to check for class imbalances.



In the bubble chart (left), each bubble is a character and the y-axis represents the lines spoken by the top 20 characters for Season 3. Some characters consistently speak fewer dialogs, we hypothesized that such characters may not present enough data points for the model to learn on. As not enough class representation in a model can produce a weak learner. In the bar charts (bottom), we show the Total words spoken by a character colored by gender and we show the overall sentiment scores for the characters. One can visually see the imbalance between classes (characters) per both these visuals (the bubble chart to the left and the total words spoken by character and gender bar chart below. Due to class imbalance, gender and sentiment scores are used as features for the supervised learning models to help identify class labels properly. For instance, we can see from the sentiment scores that most of the adults except one character, display a negative sentiment score. While the younger characters are positive. By conducting an initial EDA, such as this, we were provided with the background we needed to anticipate challenges when training our models. We approached these issues by tuning the hyperparameters along with training our models on filtered iterations of our dataset based on characters who were represented well with enough data for the model to learn efficiently along with adding adequate features.





As mentioned above, to address the class imbalance in our dataset, we essentially downsampled to train on the more highly represented data points, in this case, characters who spoke enough dialogs as well as filtering out instances where the character spoke dialog where the token length was 3 words or less, with filtering we were able to represent both genders equally as well, to train the model efficiently and help the model converge faster.

Methods & Evaluation

Logistic Regression Classifier

Logistic regression was our baseline model for classification as it is good to use to understand the strength of relationships by leveraging signals to approximate the most appropriate classification. We utilized the Logistic Regression module via the Sklearn library and split our output from our TF-IDF via Train Test Split with an 80/20 ratio. Initially, we started with default parameters, no features, n estimators set to 100, and fit our training data, then used predict with our test data. At initial runtime, we received test accuracy scores of 17.6% and train accuracy scores of 45.1%. We then started hyperparameter tuning via GridSearchCV, parameters explored were C at values of [0.01,0.1,0.5,1.0,5,10], penalty of L1 or L2, solver of ['liblinear','lbfgs','sag','saga'] with max_iter set to 250, CV equal to 2 and scoring set to accuracy. With the final tuned parameters returned of C equal to 1.0, the penalty set to L1 and the solver set to liblinear and we set our model's max_iter to 500.

After setting the ideal parameters, the model was rerun and the accuracy improved ever so slightly. We then had to go in and add features as described above such as age, gender, sentiment, season, and episode. These features immediately helped and the returned test accuracy scores jumped slightly to 39.6%. With still low accuracy test set scores, we decided to filter the data to help with any class imbalances and made sure to provide the model with adequate representations of classes for it to learn. Any lines of dialog where the tokenized length was less than 3 and characters who appeared in the series 2 times or less were filtered out and results improved once again. To truly get the model to have an enriched dataset and make the most use of our features we then only kept the Top 20 characters which had a final accuracy test score of 75.4% and training accuracy of 93%. Further details of the model, data filtering, parameters used and iterations can be found in Table 1 in the appendix.

During the process of hyperparameter tuning at various iterations, GridSearchCV took very long to run and we had to decrease the number of parameters we searched in each of our categories, ideally, we would have liked to search and test as many parameters as possible.

Random Forest Classifier

Our goal with using Random Forest for classification was to see how well the model performed on the dataset since it can easily perform the Who Said What tasks on noisy and/or highly dimensional data by making use of an ensemble of decision trees for classification. Randomized Search CV was utilized for hyperparameter tuning and was much faster than using GridSearchCV due to how Randomized Search CV evokes sampling. We used a range of high to low values (100-1000) for building the number of estimators to create an aggregate model with variance - 500 being the most ideal, a minimum sample range between 2-20 to split on at an internal node of the trees with 20 being the most ideal, and an individual decision tree to reduce overfitting and increase explainability of the tree by reducing the possible number of paths to the leaf nodes with one being most ideal [4]. Random Forest rarely overfits when using a large number of trees but because of its randomness, it can potentially leave out good features that may have high predictive power. We found this to be a particular challenge during hyper-tuning and feature selection.

With no added features and only using the TF-IDF tokenized words with no hyperparameter tuning, the model performed with a score of 19.98%. Adding character features such as gender, age group, and sentiment increased performance to 47.75%. Using the same features but filtered to drop lines of text that were 3 words or less with characters who appeared in the series 3 times or more increased accuracy to 67.10%. Use Table 2 in the appendix for further details on accuracy test scores. Multiple runs of Random Forest were assessed to tune the model and examine the highest accuracy and performance due to the model's hyperparameter sensitivity. The results are graphed in a confusion matrix (presented in visual 4-2) to show true and predicted labels for the model with parameters set to max features equal to square root and n_estimators set to a value of 100. This untuned model returned the highest test score accuracy of 92% and due to the behavior of having ideal hyperparameters returned but only producing a max score of about 84% we decided to perform a full hyperparameter analysis.

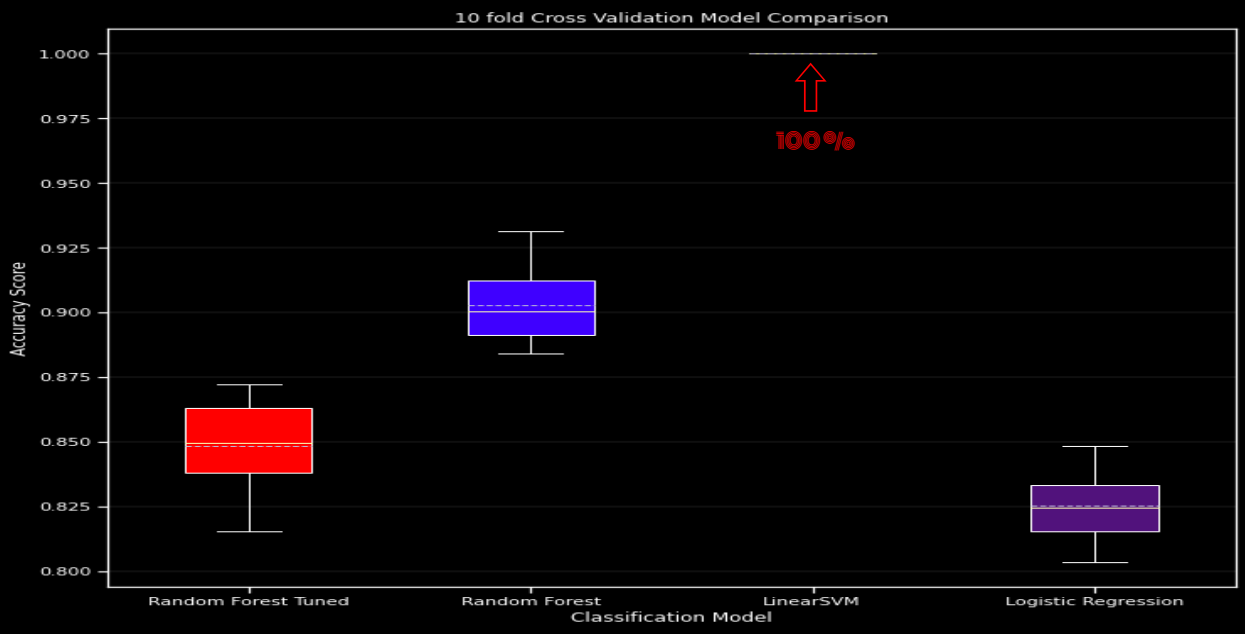
Linear SVM Classifier

We chose Linear SVM because of its known capability of high prediction accuracy, especially on a large number of features as it maximizes the decision boundary between classes. SVM also has the ability to solve for optimization by increasing the distance of decision boundary to classes (support vectors) and maximizing the number of points that are correctly classified in the training set [15]. The words from the scripts were tokenized to break text up into tokens of both words and phrases. Following this, lines of text needed were transformed into vectors to encode and leverage the power of generalization capability of the words/lines with high prediction accuracy for character name, robust to outliers. We used TF-IDF to convert the text to numerical vectorized structures and then split the corpus into a training and test data set. The training data was used to fit the model with the predictions performed on the test data at an 80/20 ratio.

We used Randomized Search CV for hyperparameter tuning with a cv set to 5. The hyperparameter tuning explored for this model were used as a valuation of "how badly" we wanted the model to properly classify so we tested with C (regularization parameter to control trade-offs between achieving a low error on the training data and minimizing the norm of the weights [9]) range of 0-1000, degree range of 0-20, gamma range between 0.001-100 and auto, and multiple kernel types (linear, rbf, poly, sigmoid).

As mentioned above we used recurring or main characters to the series overall to help with issues where one character might appear in one episode and only speak three lines in total, therefore, being underrepresented in the dataset. Unlike the other models, Linear SVM after adding features (no character filtering or line filtering to the data) test score accuracy increased substantially quicker than the other models. The final tuned hyperparameters used for the SVM model that was most ideal with a 100% accuracy score included a linear kernel, gamma value at 0.001, degree value at 20, and c value of 10. For the linear kernel, we needed to optimize the c parameter because of the small gamma. The hypertuned small gamma value indicated a large similarity radius, resulting in more points being grouped together, making it less likely to overfit.

Finalized Model Selection



We performed 10-fold cross-validation on all 3 classification models plus the fourth not fully tuned Random Forest model (since it outperformed our tuned Random Forest model due to hyperparameter sensitivity) to be confident in each model's accuracy on the training and test set scores after finding ideal hyperparameters, the addition of features and filtering down the dataset in order for the models to learn better. We needed to perform this procedure because it ensures our model's skill overall based on multiple iterations and reduces the variance of the model's performance by averaging k over different partitions. This makes it so the performance estimates provided for the comparison of our models are less sensitive to the partitioning of the data and generally result in an overall less biased/optimistic estimate of the model and can be summarized via the mean accuracy and mean variance (standard deviation) values provided by the output. As we can see, in the below table, our Linear SVM model has the best mean accuracy and zero variance for the 10-fold cross-validation results and makes a concrete determination that it was truly our best-performing model.

10 Fold Cross Validation Results		
Model Name	**Mean Accuracy**	**Std Accuracy**
LinearSVM	1	0
Logistic Regression	0.825297619	0.015048658
Random Forest	0.90297619	0.015253279
Random Forest Tuned	0.848511905	0.019863579

Logistic Regression Classifier

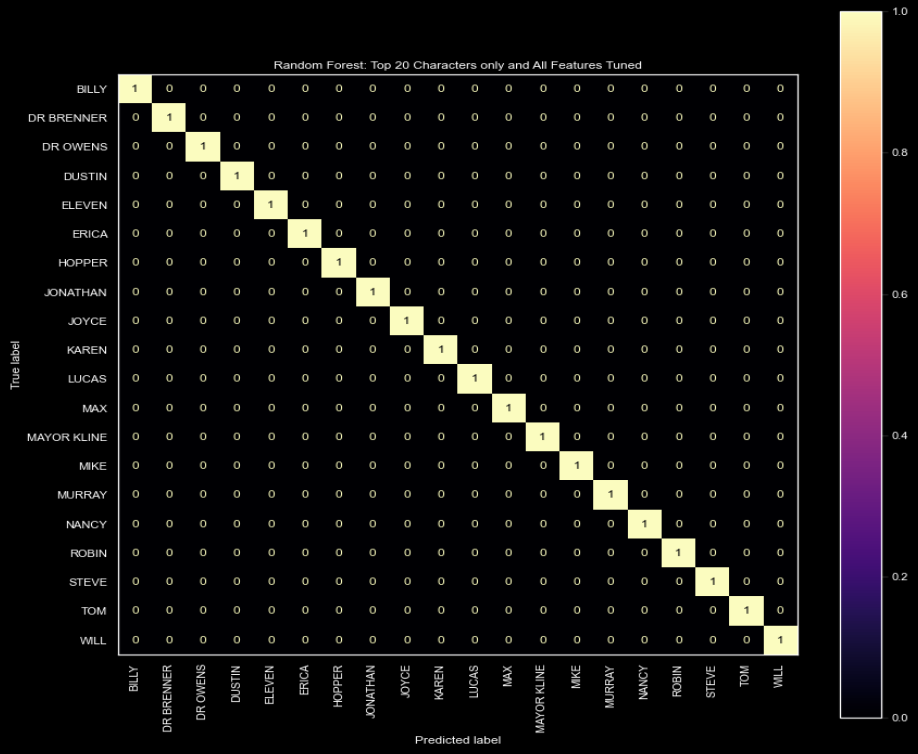
With data filtering at multiple stages and adding features during stages along with hyperparameter tuning this model performed the worst after using a finalized dataset with features and tuned parameters as you can see above it had an accuracy after 10-fold cross-validation of just 82.5%.

Random Forest Classifier

Initially, going into this project we felt the Random Forest Classifier would outperform the other models but it turns out this model is very sensitive to hyperparameter tuning and while it still performed decently with 85% accuracy on our test set after feature additions and dataset filtering along with hyperparameter tuning it was very sensitive. We performed hyperparameter analysis on this model due to its sensitivity and as it turns out even though we had specific hyperparameters suggested from the Randomized Search CV results, using just a few of those parameters and not all the others we tested together (leaving at default) values gave accuracy on test results of 92% and mean accuracy of 90%.

Linear SVM Classifier

Linear SVM outperformed all 3 models at 100% accuracy on the test set and 100% accuracy after 10-fold cross-validation. Even before fully tuning and just adding features onto the dataset along with some filtering it was outperforming both of the other models. Examining the final confusion matrix below shows the results from our final Linear SVM classifier and displays excellent results in terms of predicted label versus true label for our test dataset at 100% accuracy. We believe the high resultant test accuracy and mean accuracy score are attributed to the linear kernel which is ideal for text classification since text has many many features and are generally linearly separable, hence the performance boost of using the linear kernel [2].



Analysis of Models

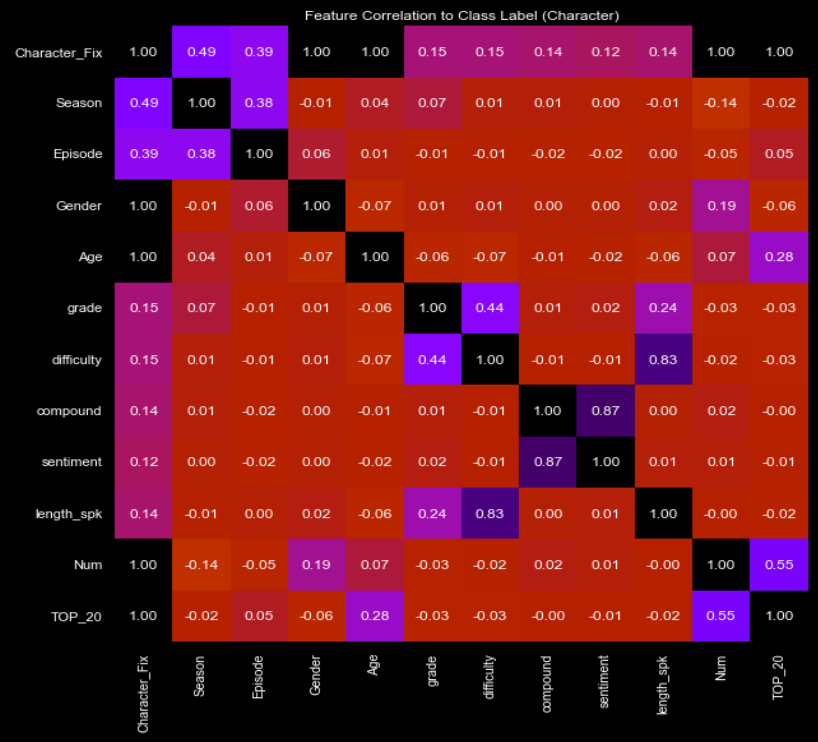
Hyperparameter Analysis

Hyper Parameter	Train Accuracy	Test Accuracy
Finalized Tuned	0.9734	0.8508
Default Setting	1.0000	0.9107
N Estimators = 500	1.0000	0.9306
Min Samples Split = 20	0.9862	0.8790
Min Samples Leaf = 1	1.0000	0.9107
Max feature = sqrt	1.0000	0.9107
Max Depth = 30	0.9979	0.8684
Criterion = gini	1.0000	0.9107
Number of Jobs = -1	1.0000	0.9107

Hyperparameters are tunable settings for the algorithm that can be adjusted for better performance. Particularly for Random Forest, hyperparameters include the number of decision trees in the forest and the number of features that a tree considers while splitting a node. Hyperparameter tuning can best be evaluated by trial and error of different combinations to see how the performance of the model improves. If we only evaluate these features on the training set, then we risk overfitting the data. In this case, we performed many iterations of the hyperparameter analysis each time using different parameter settings on the model summarized in the table (left) utilizing the ideal parameter values from our resultant tuning to analyze sensitivity and variability we saw in our results. The table (left) shows the comparison between different hyperparameters and their training and test scores. Performance increased with a smaller parameter value (less features considered when splitting at each node) by reducing variance in the structure [4] while the amount of random features considered in the split seemed to negatively impact performance due to the possibility of leaving out features with higher predictive power. The increase in features provides a better chance that good features will be included but if it is coupled with a small number of trees, we will see increased variance and a decrease in performance since features with higher predictive power may be used very little or not used at all.

Feature Analysis

To examine possible failures with our Random Forest and Logistic Regression models (Linear SVM performed overwhelmingly well with the addition of features), we looked at the correlation of features to the class labels, aka the character. A confusion matrix was used to give us false positives, false negatives, true positives, and true negatives as a way to assess errors and calculate the accuracy of our models. When examining the correlations between the characters (our labels) and features we found high correlations between being a top 20 character, season, episode, gender, age, and num (total lines spoken in the series) with the class labels of characters. This is why at various stages in our model tuning we added features to improve the learner given their correlation. Further details to see improvements in accuracy at the additions of features in stages along with filtering down the dataset to include only Top 20 characters can be seen in Tables 1, 2 and 3 in the appendix.



Failure Analysis

To further analyze failures in our models we took all the lines spoken by a character along with their added features (Age, Gender, and compound sentiment score) to calculate the cosine similarity between each top 20 character. We wanted to use this as a guide to see why the model was mislabeling some of our main characters and could not distinguish between the two. Therefore having a positive cosine similarity could explain some of the learners' confusion between characters. For logistic regression and random forest final model's confusion matrices please refer to Confusion Matrix Visual 4-1 and 4-2 in the appendix to examine the failure analysis described here. For Eleven's incorrect lines we believe this is due to Eleven and Max sharing a cosine similarity score of 0.782 (referring to Table 4 in the appendix). Hence, why, when you examine where the logistic regression failed (see Confusion Matrix Visual 4-1 and 4-2 in the appendix), Eleven was mislabeled as Max for 42% of her lines, and for Random Forest, Eleven, was mislabeled as Max 33% of the time, we conclude this to be correlated to these characters sharing similar sentiment in words spoken, well as same features both are female in the children age group and appear on screen together so their season and episode numbers are similar and they are both in the Top 20 characters group. Precision, Recall, and Support for Eleven's lines for both Random Forest and Logistic Regression can be found in Visuals 4-1_1 and 4-2_1 on page 21 in the appendix.

Another example of this is 14% of Erica's mislabeled lines were labeled as Eleven, when examining their cosine similarity it comes in at 0.568 and they are once again when we examine the shared features they are in the same age group, children, same gender and similar overall sentiment. A third example of this same scenario is Mayor Kline's lines being mislabelled as Murray, both share a cosine similarity of 0.606 and Mayor Kline compared to all other characters in Table 4 in the appendix is most related to Murray. For logistic regression, 27% of Mayor Kline's lines were mislabeled as Murray and for our random forest model, 36% of his lines spoken were mislabeled as Murray. Once again examining the shared features of the characters, they are in the same adult age group, both male and share almost exact sentiment compound scores. It would be very hard, even though we know they are very different characters, for the model to most accurately distinguish between the two given the shared features between the characters as well as being in the same seasons/episodes. The cosine similarity rating between characters along with feature analysis can be a very good tool for examining failure within your various models as it sheds light on why your model most likely misclassified a label.

Part B: Unsupervised Learning

Motivation

Reddit is an extremely popular social media platform in which users can anonymously discuss topics of interest, including TV shows such as Stranger Things. The subreddit community \rStrangerThings is used as a discussion forum where Stranger Things fans can talk about spoilers, teasers, and make general comments about the show. We wanted to look at what users were saying about the show, mainly because we are fans ourselves! Would we find relationships hidden within clusters of words and notable happenings that occurred in the show? The TV show plot had many pivotal points (sorry in advance for any spoilers) that we thought would be the focus of discussions within the Reddit community. Some notable happenings were character deaths, suspenseful events, and the progression of the plot line. Using the titles and selftext ("full post") associated with the Stranger Things subreddit, we created unsupervised learning models to examine the hidden structures in the words by utilizing Latent Dirichlet Allocation (LDA), K-Means clustering and Non Negative Matrix Factorization (NMF)). Our goal was to extract context and representation of what users were talking about and inspect groupings/topics based on the subreddit titles and selftext. As with our supervised portion, we have no stakeholders involved in this project as we are doing this out of our pure curiosity and fascination with the Netflix series.

Data Source

The dataset was collected/created using the PushShift API, a minimalist wrapper for searching public Reddit comments/submissions, to extract post submissions from the '\rStrangerThings' subreddit. We extracted all posts ever made within the subreddit via the API call. The PushShift API query returns a json file object which we were able to iterate through and gather all submissions objects via writing to a txt file while collecting a maximum of 1000 posts per call and then sleeping and only keeping if selftext was in the returned object, resulting in the final output file, posts_st.txt which has 65,607 entries. The contents of the data used can be found in our Github repository.

Unsupervised Learning Methods

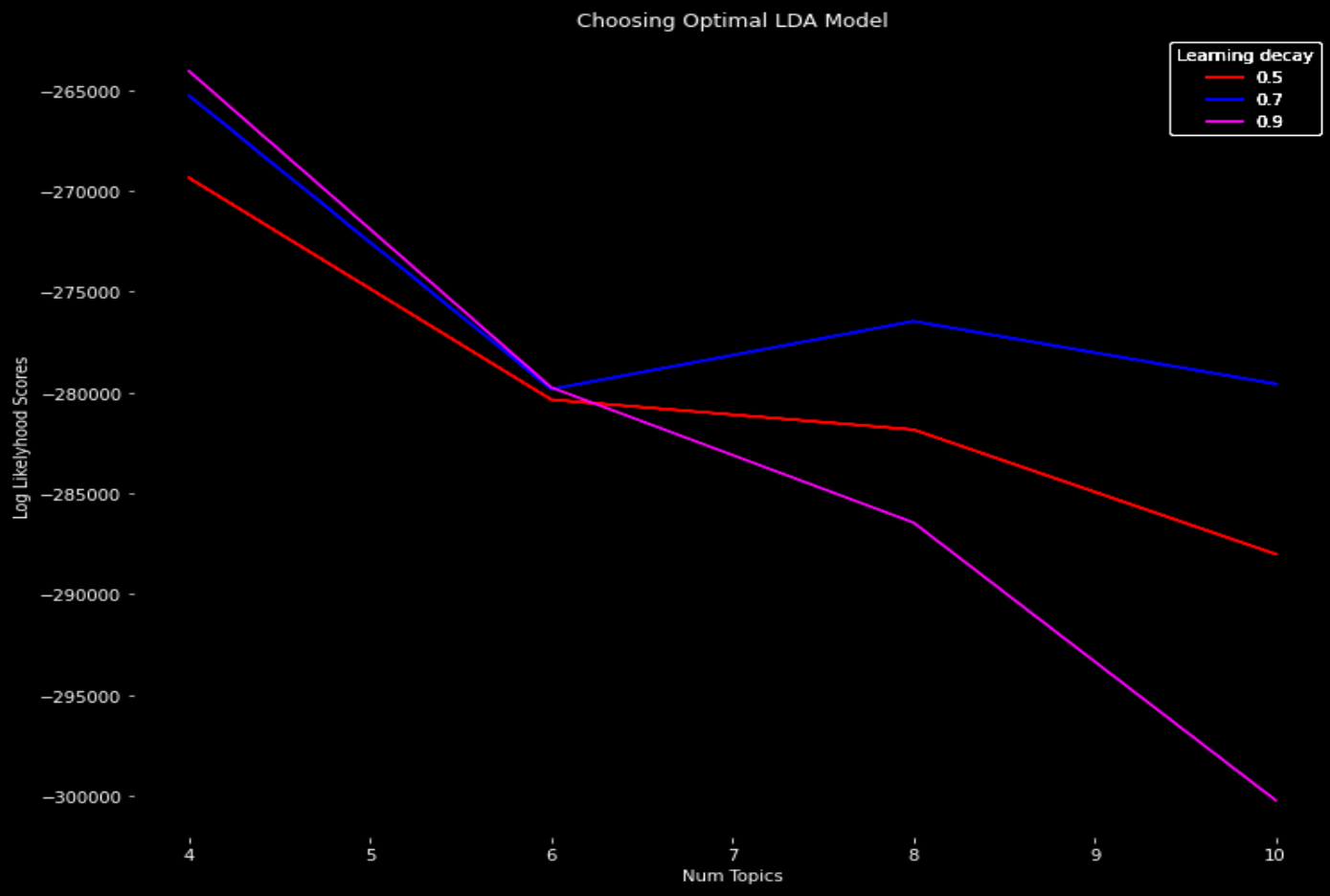
Once the data was collected from the PushShift API and into our posts_st.txt file we were able to start the cleaning process and inspection of the contents returned. When examining our output we had to delete posts where the title or the selftext were NaN, as these signified posts that either pictures only and no words or something inappropriate were removed but not designated by the subreddit moderator. Next, we removed posts where the selftext was marked as removed or deleted, these were posts deleted by the original poster or deleted by the moderator as well. We then combined the title and selftext fields into one field to create a full post that renders the title of the post and what the poster actually wrote. Then we removed punctuation, converted all characters to lowercase and removed any extra spaces along with emails or hyperlinks (via regex). Next was the removal of stop words, words less than 2 characters long, anything word that was not alpha in case and removal of any non-desired parts of speech (via Spacy's 'en_core_web_sm'). The "full posts" (title and self text) were at the same time tokenized and lemmatized (via Spacy's 'en_core_web_sm') and then joined as strings to later pass to our TF-IDF vectorizer. TF-IDF is great to feed models when the domain task is finding ideal topics/topic modeling because it measures how often a word appears in the corpus while providing the relative rarity of a term in each word in the document. To use the TF-IDF scores in our model, the words needed to be converted to a vector of numerical data through vectorization to fit our model as features. We also used the fully cleaned posts to later pass through Gensim's Word to Vec to create a full corpus dictionary to use with our coherence scorer for all models.

Using the vader_lexicon module, we applied a sentiment intensity analyzer to calculate the compound score and label the sentiment positive, neutral, or negative for each title [8]. Due to memory issues, we then decided to only use posts with a created date from July 4th, 2019, when the 3rd season of Stranger Things was released, to October 11th, 2022, which is roughly 6 months after the 4th season of Stranger Things was released. The last step in achieving fully clean and usable data was to remove full posts where the token length after cleaning and removing punctuation, removal of stop words, words less than 2 characters and non-desired parts of speech (via Spacy) with a token length of 5 or less. The titles and selftext were the most important variables for finding clusters and as combined full posts resulted in a dataframe containing a total of 30,660 records. We employed and explored 3 different unsupervised learning methods to find ideal topics and examine the topics within the Stranger Things subreddit. Each model used was picked based on different strengths (and weaknesses) along scoring metrics which we could utilize along the way.

LDA (Latent Dirichlet Allocation)

Latent Dirichlet Allocation (LDA) assumes each topic is a mixture over an underlying set of words and each full post is a mixture over a set of topic probabilities. We used LDA via Sklearn and the features we fit the model with were the cleaned and transformed full posts via TF-IDF (as mentioned previously in the section above). Once we had the TF-IDF vectors, we fit with TF-IDF and initially set at 10 components and base/default parameters to see what it would produce, which it did well. The topics made sense overall but to find the ideal number of topics and hyperparameters we used GridSearchCV (cv at default value of 5). We examined the mean log likelihood score along with the mean perplexity score from the various cross-validation splits compared to the learning decay rate to find the ideal number of topics (components). This resulted in an ideal number of topics (components) returned of 4 and as the visual below shows it is best used at a learning decay of 0.9 to assure the least negative log likelihood score along with the highest mean perplexity score. Further details of the various split and mean scores along with time scores can be seen in Table 5 in the appendices below the report.

The biggest hurdle we had to overcome was memory issues since we had 30,660 full posts (documents) and it takes a while to run grid search on various hyperparameters, we could only wait for it to run and there is nothing really to do about overcoming time. For memory issues when ended up decreasing the number of hyperparameters searched so less iterations were run by examining components [4,6,8,10] and learning decay of [0.5, 0.7, 0.9] and default parameters except for batch size set to 5000, learning method set to online and n_jobs set to -1 to improve efficiency. One attribute of why the model might have returned such a lower number of ideal topics than our other 2 models is that LDA assumes each document has multiple topics and is best suited when used on long texts such as books, essays or long articles so this perhaps wasn't the best model given the domain and the varied length of the full posts.When examining the top 20 words from the returned tuned model set to 4 components the topic words are very specific to the Stranger Things Fandom and one would have to have watched the entire series along with being familiar with the various social media outlets surrounding the show for them to make sense since they relate directly to specific characters, jokes, memes, theories, or popular scenes from the show that became a craze on Tik Tok.

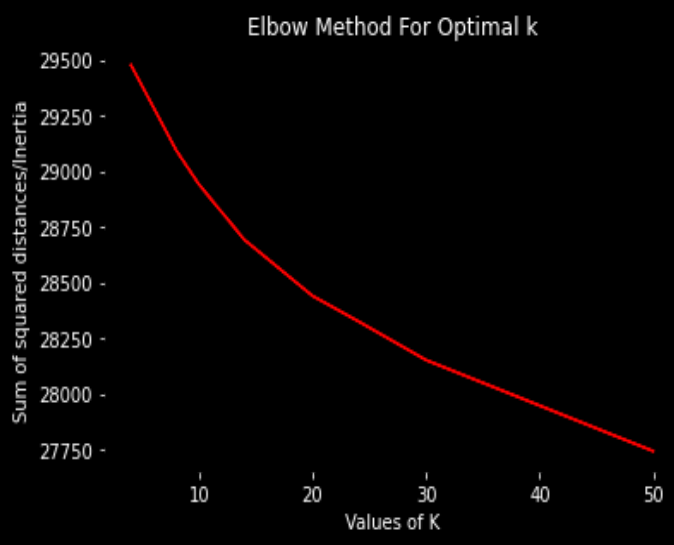


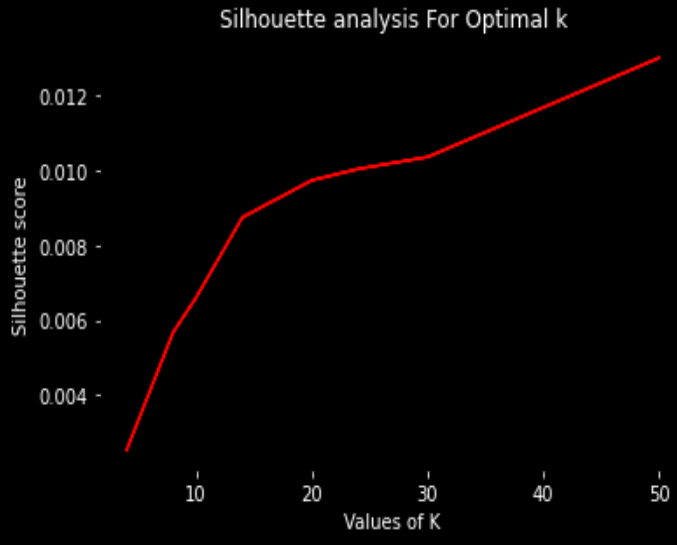
K-Means Clustering

We chose K-Means clustering as a model to explore for topic modeling since it is scalable in regard to algorithm complexity. K-Means clustering divides the full post of words into non-overlapping groups so that no one word can be included in more than one cluster. We used the Sklearn module to perform K-Means and fitted the model with features from the fully cleaned TF-IDF vectorizer of our full post. Since K-means requires that we specify the number of clusters (k), we initially used 10 clusters (n_clusters) for our model where k centroids equal to k clusters are randomly selected to represent data points at the center of each cluster. Initialization was performed to prevent any converging of the different cluster assignments using the k-means++ technique (init) to speed up convergence with a maximum of 1000 iterations for each initialization, 1000 mini batches for faster computation (batch_size), and 1000 samples to randomly sample for speeding up the initialization (init_size).

A challenge we incurred is the utilizing elbow method as well as the silhouette score to find optimal value for k. The elbow method does not distinguish a great k this could be due to the data coming all from the same subreddit so it is not exhibiting well-separated clusters. Also, this means we had to use the silhouette score as well which will help determine an ideal k. The silhouette value for each point is a measure of how similar that point is to points in its own cluster compared to points in other clusters, and ranges from -1 to +1. The best value is 1 and the worst value is -1 for silhouette scores. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar [11]. At first glance this we thought was an error in our process but after researching we realized it would make sense that there are overlapping clusters since the data stems from the same subreddit, hence have silhouette scores around 0. By examining both plots closely we felt like there is a slight bend around 14 and one around, 20 and 30 for the silhouette score which was faintly mimicked on the elbow plot below.

Using base default hyperparameters built into K-Means we wrote a defined function that iterated our 3 values for k we wanted to validate [14, 20 and 30]. Within this function we ran each model with said “k” value and return metrics of the model for time, inertia, homogeneity score, completeness, v-measure, adjusted rand score, adjusted mutual information score and our silhouette score. Based on the returned metrics we decided to use 20 clusters for the model since it returned the highest silhouette score at 0.707 and came in second for homogeneity, but still very low at 0.008, since our data does contain full posts from only one class; the topics overlap a bit since they surround the show in general. For details of returned results please see Table 6 in the appendix.

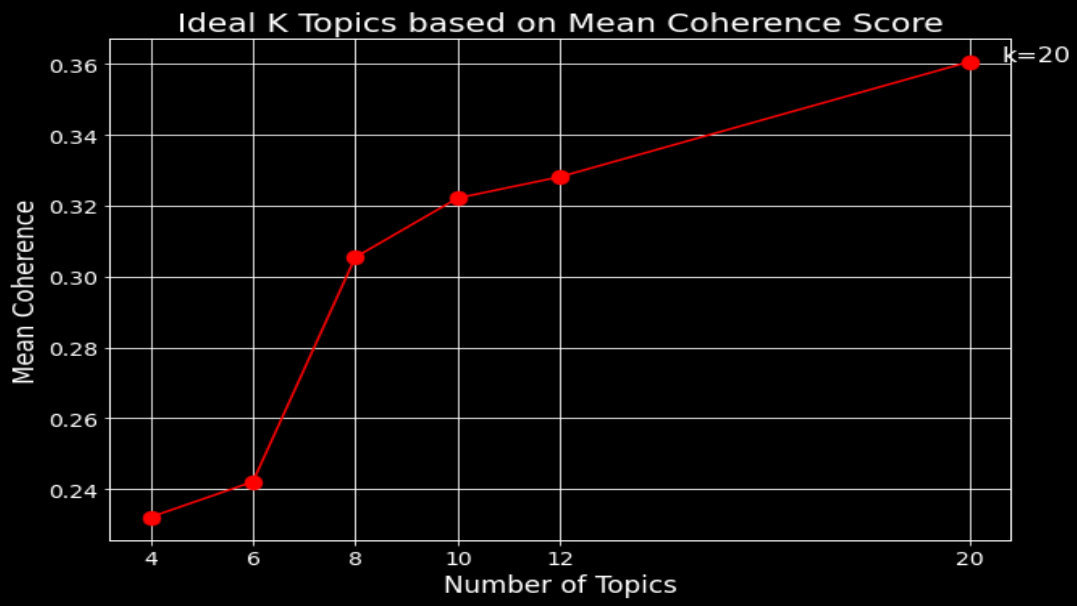




NMF (Non Negative Matrix Factorization

Non Negative Matrix Factorization (NMF) is a linear algebra-based model that factors high dimensional vectors into their low dimensional representation. It is similar to PCA in the fact that NMF takes advantage of the fact that the vectors are always non-negative. To use NMF we use Sklearn and our fully cleaned and lemmatized full posts via the vectorized output from our TF-IDF for features to fit the model. We first tried just using 10 topics and although the results looked great we wanted to be sure this was ideal.

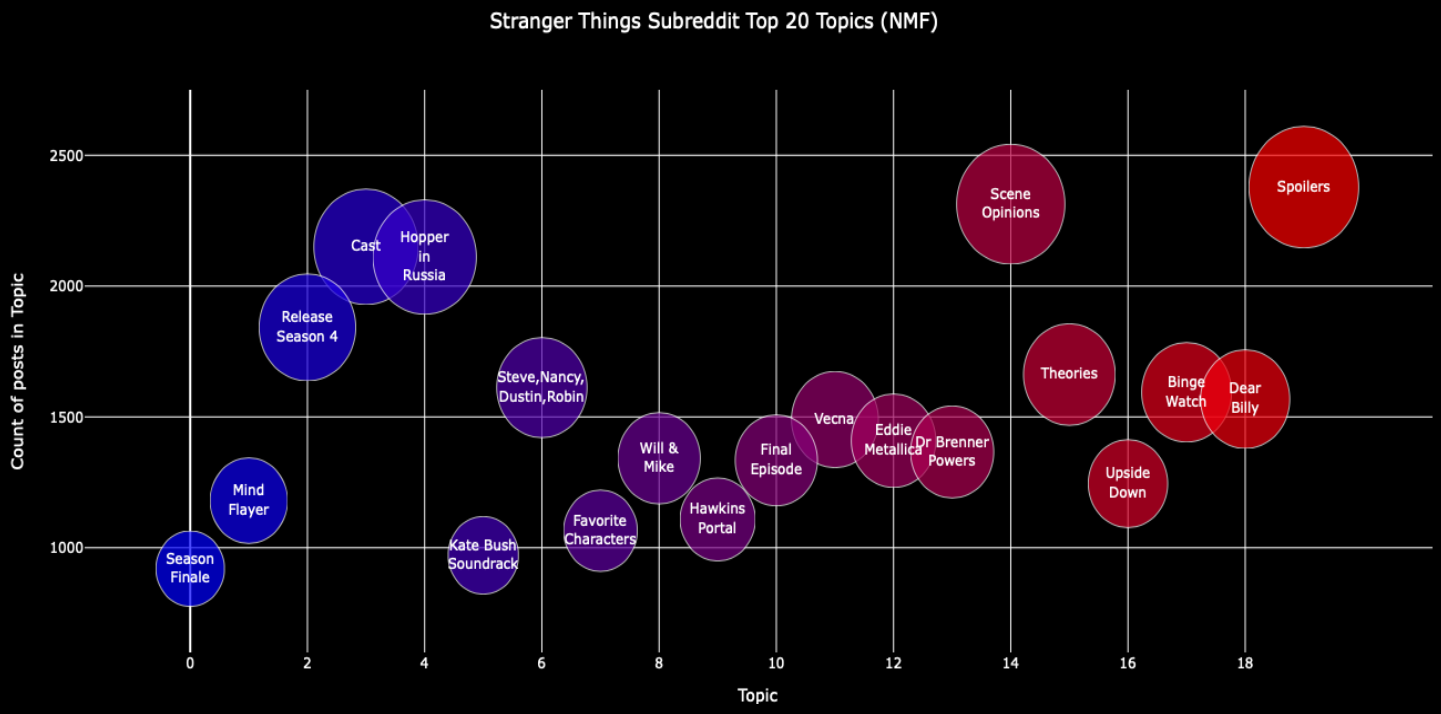
We did not have much memory or time issues with NMF at all but one issue we came across is that due to the nature of NMF, Randomized Search CV and Grid Search CV do not work with NMF for hyperparameter tuning since it is listed in Sklearn as a decomposition module similar to PCA, hence there is no predefined metric/score to use within Sklearn. Therefore we had to set base parameters of max iter at 2000 and init set to "nndsvd" after researching and then used a for loop to iterate through different values of k in the range of [4, 6, 8, 10, 12, 20] and fit the model with our vectorized results from our TF-IDF. We then wrote out our own defined functions to calculate the coherence score of each iteration of k and used a value of k which returned the highest coherence score. The number of components (k) with our researched parameters stated above returned the ideal k to be 20 with a coherence score of 0.36 as shown below in the line chart of "Ideal K Topics based on Mean Coherence Score". Further details of the coherence score of each k iteration can be found in Table 7 in the below appendix.



Unsupervised Evaluation

To determine which of our 3 models produced the ideal and best topics we used coherence scores based on our models ideal parameters and topics. The coherence score measures the human readability of the topics. By tokenizing the words and converting the tokenized objects into a corpus and dictionary using the Gensim word 2 vec module, we were able to utilize this along with the our TF-IDF and the term rankings for the top 10 terms from each k topics in a model to manually compute via a multiple defined functions and for loops the coherence score per model. The 10 top words from a topic of k components are the words with the highest probability of belonging to a particular topic and the coherence score measures how similar these words are to each other. The results from the table below show that our NMF model outperformed both other models with a coherence score of 0.3607 when ideal clusters are set to 20 and max_iter is set to 2000 and init set to "nndsvd". We believe this is due to the matrix factorization and multivariate analysis technique within the model; it also works best with short texts such as those found in social media. Some great attributes about NMF is its much faster than LDA or K-Means in terms of time and computation and it assumes a deterministic framework unlike LDA or K-Means which are non deterministic when run multiple times on the same data. NMF also calculates how well each document fits each topic and also factors in that a document can have multiple topics. Given our domain of subreddit full posts it makes sense that NMF was our ideal model.

Unsupervised Model Evaluation: Coherence Score w/ Tuned Parameters and Ideal K		
LDA: K = 4	**Kmeans: K = 20**	**NMF: K = 20**
0.1864	**0.2432**	**0.3607**



The finalized results from using our winning model, NMF, which had the highest coherence scores after comparing all models with their ideal hyperparameters and number of k-components resulted in the visualization above. We tagged the full posts to their highest weighted topic (cluster) number and used the top 20 words from each topic to label the topic number with an interpretable and readable topic name/association to give insight and display the results of NMF in the above bubble chart. An interesting relationship we saw during the unsupervised learning portion of our project is that all of the topic clusters regardless of the model shared connections around and displayed topics centered around specific happenings such as scenes, fan theories, favorite characters and the soundtrack. With that said, interpreting the results based on how many components fed does require some in-depth knowledge of the show or its network of various fandom domains to feel comfortable and see that the topics returned are indeed topics when the number of components are low as the resulting top words become very specific.

With topic modeling it can be hard to get various scoring metrics of decent values when the topics already fall under the umbrella of one topic, as ours does. All of our posts are from just the Stranger Things subreddit so technically they are all related to each other to start out with versus if we had just pulled all top posts on Reddit regardless of the subreddit for a given time period it might have been easier to see better scoring and densely formed topics. We initially thought our pipeline and hyperparameter tuning failed due to not so great metrics but we overcame the worry by researching topic modeling within given sub domains and felt better that this is a common occurrence. Nonetheless we are pleased with the results.

Discussion

Supervised

Cleaning and utilizing PDFs as a data source for the supervised learning models is very complicated if retrieving other data besides an embedded table given the format. Also, utilizing GridSearchCV when looking at multiple values of multiple hyperparameters is time and computationally expensive versus the trade-off of RandomizedSearchCV being much quicker given the sampling aspect but with less guarantee, they are indeed the ideal parameters since it is sampling based. We would like to extend our Linear SVM model into the future by testing it out as more teleplays for the newer episodes from Season 4 become available and corrections are made to past Season 2 and Season 1 episodes to see how well the model performs and still use the same feature additions to the data. We believe that it would still perform with high accuracy.

Unsupervised

Ideally when hyperparameter tuning our unsupervised learning models it became a time and memory issue where the tuner would run for hours on end or we would get memory issue alerts but we were able to overcome them. We did not expect such issues since it is 30,660 full posts in total and they all vary in length. We initially tried using Gensim library for LDA but there is no built in hyperparameter tuning within the library so iterations are ran via for loops for any parameters one would want to test out and that is where we incurred the most memory issues but luckily we were able to utilize Sklearn and Grid Search for LDA which saved on compute power. This taught us quite a bit on how computationally expensive hyperparameter tuning is. We also learned there are millions of different ways to clean text data and one way isn't necessarily better than the other, especially if you have a targeted goal of the cleaning given the domain. For example, using Regex versus built in library packages for punctuation removal or the removal of stop words; one could use a built-in module in a library or one could create their own stop words via a list for removal. We also learned that DBScan and HCA did not suit our problem type and were shelved due to difficulties surrounding the results and getting the model going properly. We believe they are still good clustering algorithms but not for problem types involving text with no labels or no designated number of clusters. The most surprising results from the unsupervised learning models we used were how aligned users were in their discussions, although the number of topics varied they all seemed to touch on main scenes, theories or music involved and surrounding the series. We would like to extend our solution into the future and delve more into Season 5 theories along with comparing topics and documents surrounding Season 5 once it is released in the future to what ends up happening in the season. Also given more time and resources (ram & cores) we would love to be more specific in tuning the 3 models.

Ethical Considerations

What ethical issues could arise in providing a solution to Part A, and how could you address them?
We do not have any ethical considerations for our supervised learning portion as it was scripts/teleplays surrounding a Netflix series and a basic text classification problem type.

What ethical issues could arise in providing a solution to Part B, and how could you address them?
Ethical considerations for how we prepared our unsupervised models can be directly attributed to the ethical use of social media data since Reddit is one of the largest social media platforms. During the collection process of gathering titles and selftext, informed consent was not provided for all users who had submissions to the \rStrangeThings subreddit used in our models so users will not know to what extent their data was accessed and analyzed for the purpose of our analysis. In more traditional research approaches, informed consent is acquired with awareness of how participation and data will be used but the terms and conditions of many social media platforms are typically agreed upon without having been read and fully understood. The agreements users agree to will typically include clauses on the accessing and re-use of data by third parties [3] such as student researchers.

Anonymity is another ethical consideration which can be addressed by Reddit maintaining a platform of anonymity. Users can choose their usernames and reveal as much or as little information about themselves by interacting within different subreddits of interest. The analyses performed did not use user names, profiles, or history - only submissions to the \rStrangerThings subreddit but the data frames are stored in the Github repository which maintains user names with titles and selftext submissions. It may be unlikely that a user will be identified through our work but it is important to consider that given the amount of data and aggregation required for the analyses, anonymization becomes more complex and difficult for individual data extracts (such as submissions) when the same models and methodologies are constantly reproduced in publications and articles.

Statement of Work

Jacqueline Skunda:

- Conceptualized the overall analyses and the easiest path to obtaining the dataset and creating the supervised training data set
- Logistic Regression & LinearSVM for the scripts/teleplays
- LDA (Latent Dirichlet Allocation) and K-means clustering for unsupervised subreddit posts/selftext
- Hyper Parameter Tuning
- Failure and Feature Analysis

Tamara Qawasmeh:

- Gathering of SubReddit titles/selftext
- Cleaning of posts/selftext along with vectorizer
- Checking of distribution of self text/comments associated posts along with word frequency for the Subreddit data
- Use of NER and on the subreddit data along with creation of code to gather sentiment scores via Vader Lexicon.
- NMF for unsupervised subreddit posts/comments

Malini Varadarajan:

- PM Lead/Coordinator
- EDA of scripts/teleplays; checking of distribution of Characters for teleplays and lines spoken/word frequency
- Random Forest Classifier for scripts/teleplays
- Hyper Parameter Analysis
- Visualizations for EDA and report writing

All: Commits/changes/notebooks to Github: Github Repo for Milestone 2

- Data Extraction, Cleaning and Manipulation & Feature Engineering
- Synthesizing data and drawing conclusion

Citations

1. Stranger Things • Screenplay. 8FLiX. https://8flix.com/
2. Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin. "A practical guide to support vector classification." (2003): 1396-1400.
3. Townsend, Leanne, and Claire Wallace. "Social media research: A guide to ethics." University of Aberdeen 1 (2016): 16.
4. Thorn, J. (2021, December 16). Random Forest: Hyperparameters and how to fine-tune them. Medium. Retrieved from https://towardsdatascience.com/random-forest-hyperparameters-and-how-to-fine-tune-them-17aee785ee0d
5. sklearn.ensemble.RandomForestClassifier. (n.d.). Scikit-learn. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
6. sklearn.svm.LinearSVC. (n.d.). Scikit-learn. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
7. sklearn.linear_model.LogisticRegression. (n.d.). Scikit-learn. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
8. Vader Lexicon: Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.
9. What is the influence of C in SVMs with linear kernel? (2012, June 23). Cross Validated. Retrieved from https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel/159051
10. Selecting the number of clusters with silhouette analysis on KMeans clustering. (n.d.). Scikit-learn. Retrieved from https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html
11. Selecting the number of clusters with silhouette analysis on KMeans clustering. (n.d.). Scikit-learn. Retrieved from https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html
12. Selecting the number of clusters with silhouette analysis on KMeans clustering. (n.d.). Scikit-learn. Retrieved from https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html
13. Stranger Things Wiki | Fandom. (n.d.). Retrieved from https://strangerthings.fandom.com/wiki/Stranger_Things_Wiki
14. GitHub - pushshift/api: Pushshift API. (n.d.). GitHub. Retrieved from https://github.com/pushshift/api
15. Yıldırım, S. (2021, December 16). SVM Hyperparameters Explained with Visualizations - Towards Data Science. Medium. https://towardsdatascience.com/svm-hyperparameters-explained-with-visualizations-143e48cb701b

APPENDIX

Table 1 - Logistic Regression: Model Analysis and results per feature set dataset addition and filters (Failure Analysis)

Dataset Fed to Model	C	Solver	Penalty	Max Iter	N Jobs	Accuracy Train Score	Accuracy Test Score
All Characters no features	None	liblinear	None	500	-	45.10%	17.60%
All Characters with featues	None	liblinear	None	500	-1	71.10%	39.60%
All Characters with all featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	0.1	saga	L1	1000	-1	82.7%%	70.04%
All Characters with featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	0.1	liblinear	L1	500	-1	76.60%	70.04%
Top 20 Characters only with featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	0.1	saga	L1	500	-1	59.70%	45.80%
Top 20 Characters only with featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	1.0	liblinear	L1	1000	-1	93.20%	75.40%
All Characters with featues & Appeared more than 2 times in the series & line spoken longer than 3 tokens & Top 20 as a features	1.0	liblinear	L1	500	-1	92.00%	67.10%
Grid Search done on C': [0.01,0.1,0.5,1.0,5,10], 'penalty':['l1','l2], 'solver':['liblinear','lbfgs','saga'], with score = accuracy and max iter = 250 and cv = 5							
*** class weight made all of the results worse off so it was removed							

Table 2 - Random Forest: Model Analysis and results per feature set dataset addition and filters

Dataset Fed to Model	Criterion	Max Depth	Max Features	N Estimators	N Jobs	Min Samples Split	Min Samples Leaf	Accuracy Train Score	Accuracy Test Score
All Characters no features	-	-	-	100	-	-	-	45.10%	19.98%
All Characters with featues	gini	30	sqrt	500	-1	20	1	71.3%%	47.75%%
All Characters with all featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	gini	30	sqrt	500	-1	20	1	93.90%	78.50%
All Characters with featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	gini	30	sqrt	500	-1	20	1	93.90%	78.50%
Top 20 Characters only with featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	-	-	sqrt	100	-	-	-	100.00%	92.09%
Top 20 Characters only with featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	gini	30	sqrt	500	-1	20	1	96.70%	84.40%

Table 3 - Linear SVM:: Model Analysis and results per feature set dataset addition and filters

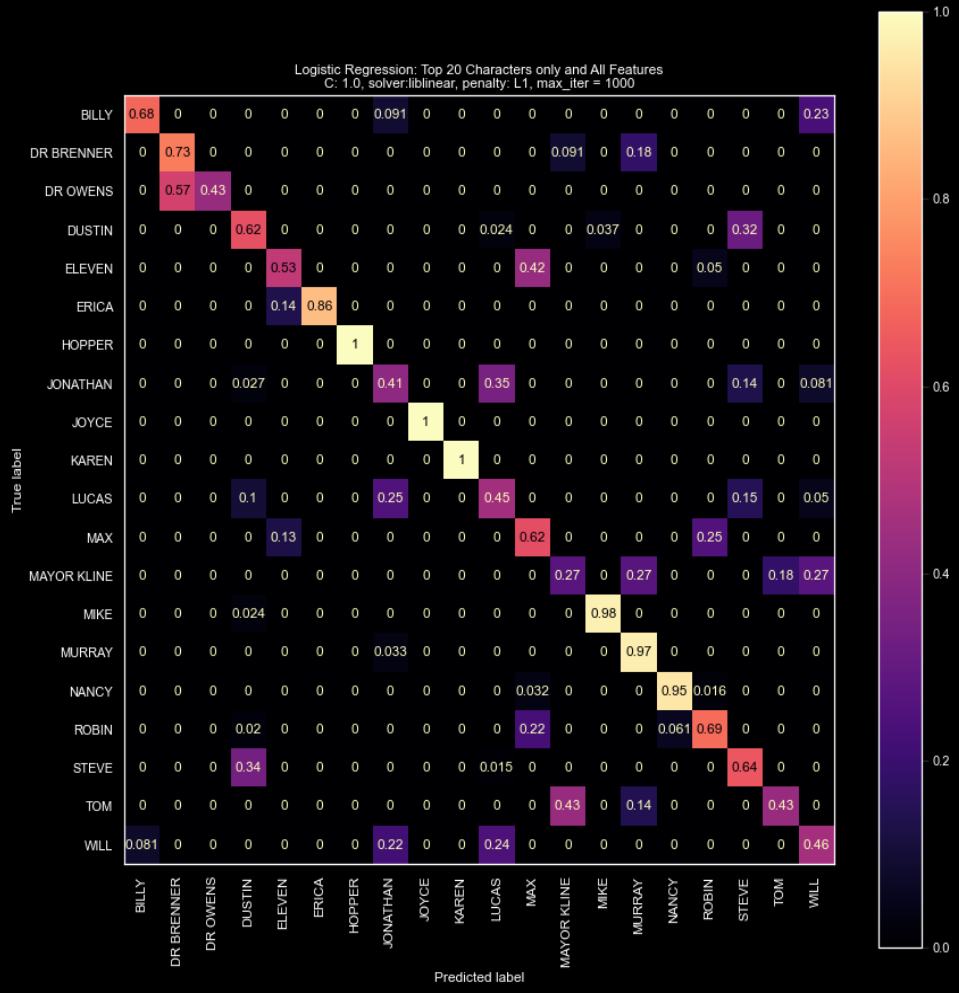
Dataset Fed to Model	C	Degree	Gamma	Kernel	Accuracy Train Score	Accuracy Test Score
All Characters no features	-	3	auto	linear	-	19.60%
All Characters with featues	1.0	30	0.1	linear	61.30%	46.10%
All Characters with all featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	1.0	30	0.1	linear	99.80%	99.20%
All Characters with featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	1.0	30	0.1	linear	99.80%	99.20%
Top 20 Characters only with featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	1.0	30	0.1	linear	100.00%	100.00%
Top 20 Characters only with featues & Appeared more than 2 times in the series & line spoke longer than 3 tokens	10.0	20	0.01	linear	100.00%	100.00%

Table 4 - Failure Analysis 1 via Similarity Scores between Characters and their Features

Character	BILLY	DUSTIN	ELEVEN	ERICA	HOPPER	JONATHAN	JOYCE	KAREN	LUCAS	MAX	MAYOR KLINE	MIKE	MURRAY	NANCY	ROBIN	STEVE	TOM	WILL
BILLY	1.000	-0.414	-0.262	-0.372	-0.386	0.601	-0.412	0.129	0.617	-0.290	-0.434	0.694	-0.450	0.598	-0.417	-0.401	0.613	0.616
DUSTIN	-0.414	1.000	0.592	0.666	0.643	-0.396	0.614	0.230	-0.330	0.643	0.576	-0.284	0.627	-0.362	0.730	0.764	-0.446	-0.359
ELEVEN	-0.262	0.592	1.000	0.568	0.676	-0.383	0.641	0.284	-0.328	0.782	0.573	-0.218	0.567	-0.341	0.573	0.598	-0.414	-0.343
ERICA	-0.372	0.666	0.568	1.000	0.614	-0.376	0.593	0.229	-0.337	0.625	0.562	-0.305	0.563	-0.343	0.664	0.686	-0.422	-0.354
HOPPER	-0.386	0.643	0.676	0.614	1.000	-0.378	0.867	0.242	-0.359	0.671	0.683	-0.282	0.727	-0.339	0.637	0.694	-0.437	-0.358
JONATHAN	0.601	-0.396	-0.383	-0.376	-0.378	1.000	-0.378	0.009	0.640	-0.364	-0.426	0.674	-0.438	0.779	-0.401	-0.356	0.648	0.619
JOYCE	-0.412	0.614	0.641	0.593	0.867	-0.378	1.000	0.236	-0.375	0.638	0.650	-0.322	0.696	-0.351	0.623	0.671	-0.443	-0.381
KAREN	0.129	0.230	0.284	0.229	0.242	0.009	0.236	1.000	0.005	0.289	0.207	0.081	0.188	0.021	0.234	0.257	-0.031	0.015
LUCAS	0.617	-0.330	-0.328	-0.337	-0.359	0.640	-0.375	0.005	1.000	-0.277	-0.423	0.724	-0.417	0.652	-0.367	-0.332	0.560	0.649
MAX	-0.290	0.643	0.782	0.625	0.671	-0.364	0.638	0.289	-0.277	1.000	0.579	-0.186	0.571	-0.314	0.622	0.656	-0.406	-0.298
MAYOR KLINE	-0.434	0.576	0.573	0.562	0.683	-0.426	0.650	0.207	-0.423	0.579	1.000	-0.396	0.606	-0.402	0.568	0.608	-0.462	-0.420
MIKE	0.694	-0.284	-0.218	-0.305	-0.282	0.674	-0.322	0.081	0.724	-0.186	-0.396	1.000	-0.393	0.685	-0.341	-0.283	0.582	0.739
MURRAY	-0.450	0.627	0.567	0.563	0.727	-0.438	0.696	0.188	-0.417	0.571	0.606	-0.393	1.000	-0.410	0.577	0.606	-0.467	-0.430
NANCY	0.598	-0.362	-0.341	-0.343	-0.339	0.779	-0.351	0.021	0.652	-0.314	-0.402	0.685	-0.410	1.000	-0.366	-0.321	0.673	0.627
ROBIN	-0.417	0.730	0.573	0.664	0.637	-0.401	0.623	0.234	-0.367	0.622	0.568	-0.341	0.577	-0.366	1.000	0.811	-0.448	-0.379
STEVE	-0.401	0.764	0.598	0.686	0.694	-0.356	0.671	0.257	-0.332	0.656	0.608	-0.283	0.606	-0.321	0.811	1.000	-0.437	-0.338
TOM	0.613	-0.446	-0.414	-0.422	-0.437	0.648	-0.443	-0.031	0.560	-0.406	-0.462	0.582	-0.467	0.673	-0.448	-0.437	1.000	0.560
WILL	0.616	-0.359	-0.343	-0.354	-0.358	0.619	-0.381	0.015	0.649	-0.298	-0.420	0.739	-0.430	0.627	-0.379	-0.338	0.560	1.000

Visual 4-1 (Logistic Regression) and Visual 4-2 (Random Forest) - Confusion Matrices For Failure Analysis when examining Logistic Regression and Random Forest (Linear SVM results are in the report but it performed at 100% accuracy so we are not using it for failure analysis.

Logistic Regression Matrix Final(Visual 4-1)



Random Forest Matrix Final(Visual 4-2)

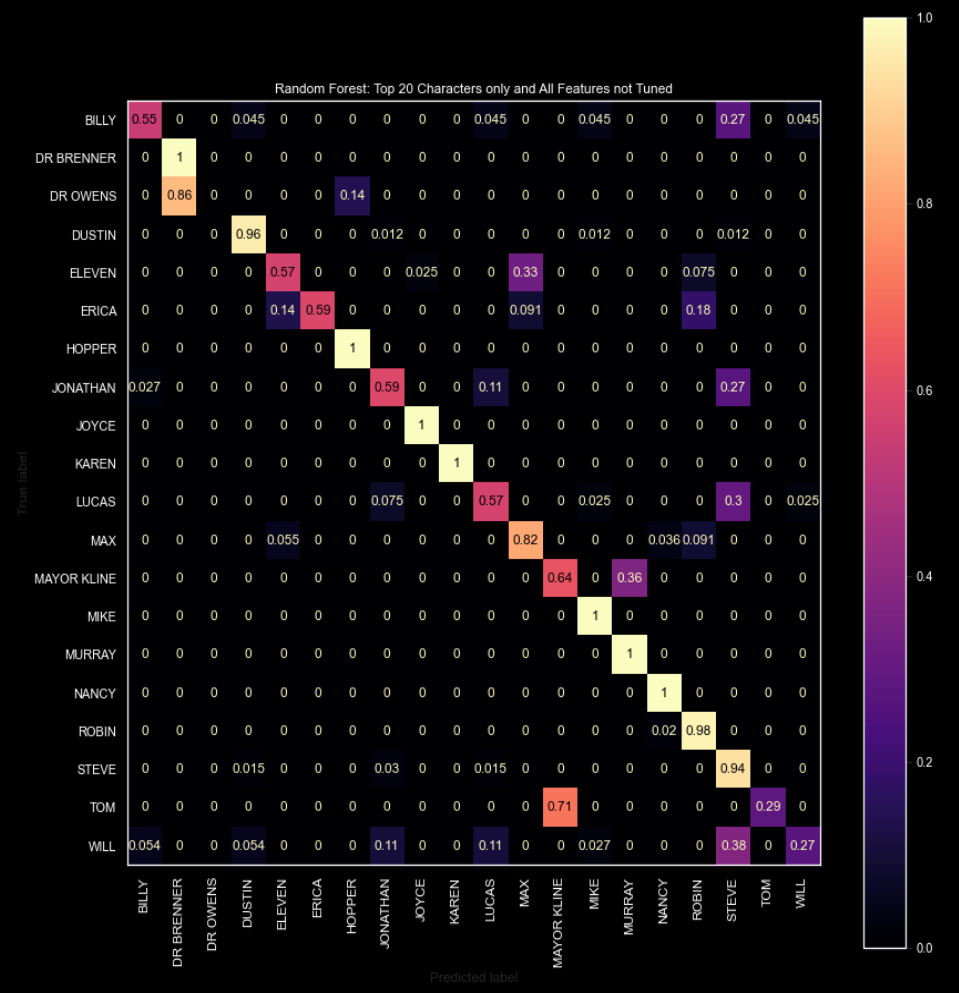


Table 4, Visual 4-1 and Visual 4-2 Failure Analysis Continued: Precision, Recall and Support for Logistic Regression and Random Forest finalized models to supplement confusion matrices (Visuals 4-1 && 4-2)

Logistic Regression Precision, Recall, Support Final(Visual 4-1_1)

Classification report of imbalanced logistic regression	precision	recall	f1-score	support
BILLY	0.83	0.68	0.75	22
DR BRENNER	0.67	0.73	0.70	11
DR OWENS	1.00	0.43	0.60	7
DUSTIN	0.62	0.62	0.62	82
ELEVEN	0.68	0.53	0.59	40
ERICA	1.00	0.86	0.93	22
HOPPER	1.00	1.00	1.00	104
JONATHAN	0.42	0.41	0.41	37
JOYCE	1.00	1.00	1.00	64
KAREN	1.00	1.00	1.00	19
LUCAS	0.42	0.45	0.43	40
MAX	0.53	0.62	0.57	55
MAYOR KLINE	0.43	0.27	0.33	11
MIKE	0.97	0.98	0.97	85
MURRAY	0.83	0.97	0.89	30
NANCY	0.95	0.95	0.95	62
ROBIN	0.67	0.69	0.68	49
STEVE	0.54	0.64	0.59	67
TOM	0.60	0.43	0.50	7
WILL	0.57	0.46	0.51	37
accuracy			0.75	851
macro avg	0.74	0.69	0.70	851
weighted avg	0.76	0.75	0.75	851

Random Forest Precision, Recall, Support Final(Visual 4-2_1)

Classification report of imbalanced Random Forest Classifier not Tuned	precision	recall	f1-score	support
BILLY	0.80	0.55	0.65	22
DR BRENNER	0.65	1.00	0.79	11
DR OWENS	0.00	0.00	0.00	7
DUSTIN	0.95	0.96	0.96	82
ELEVEN	0.79	0.57	0.67	40
ERICA	1.00	0.59	0.74	22
HOPPER	0.99	1.00	1.00	104
JONATHAN	0.69	0.59	0.64	37
JOYCE	0.98	1.00	0.99	64
KAREN	1.00	1.00	1.00	19
LUCAS	0.70	0.57	0.63	40
MAX	0.75	0.82	0.78	55
MAYOR KLINE	0.58	0.64	0.61	11
MIKE	0.96	1.00	0.98	85
MURRAY	0.88	1.00	0.94	30
NANCY	0.95	1.00	0.98	62
ROBIN	0.80	0.98	0.88	49
STEVE	0.59	0.94	0.73	67
TOM	1.00	0.29	0.44	7
WILL	0.83	0.27	0.41	37
accuracy			0.85	851
macro avg	0.80	0.74	0.74	851
weighted avg	0.85	0.85	0.84	851

Table 5 - LDA DETAILED HYPER PARAMETER TUNING RESULTS

number component	learning decay	mean fit time	std fit time	mean score time	std score time	split0 test score	split1 test score	split2 test score	split3 test score	split4 test score	mean test score	std test score	rank test score
4	0.9	19.41	1.08	0.61	0.01	-269002.32	-273225.04	-261837.85	-253624.77	-262683.85	-264074.77	6695.51	1
4	0.7	40.04	13.10	1.57	0.71	-268910.64	-273712.72	-263540.83	-255503.44	-264781.87	-265289.90	6052.20	2
4	0.5	33.81	3.60	1.07	0.22	-275755.02	-277224.78	-266714.84	-259172.55	-267978.10	-269369.06	6564.17	3
8	0.7	25.40	3.84	1.00	0.48	-287448.62	-286541.12	-271417.36	-263728.96	-273214.28	-276470.07	9169.74	4
10	0.7	25.47	5.31	0.73	0.12	-283349.67	-290764.25	-276985.04	-268804.41	-277977.53	-279576.18	7276.17	5
6	0.9	22.18	3.41	0.73	0.12	-278703.78	-297254.51	-278015.32	-267257.32	-277531.30	-279752.45	9710.83	6
6	0.7	32.62	6.68	0.93	0.17	-274723.00	-293209.65	-280270.27	-270721.94	-280260.98	-279837.17	7595.17	7
6	0.5	27.30	2.24	0.86	0.09	-279589.81	-291793.22	-280654.33	-269960.50	-279786.83	-280356.94	6926.42	8
8	0.5	28.71	5.19	1.14	0.30	-288126.70	-292544.59	-278469.39	-270263.76	-279816.92	-281844.28	7791.97	9
8	0.9	18.83	2.03	0.69	0.06	-296843.72	-295062.51	-282428.74	-274296.67	-283630.62	-286452.45	8414.60	10
10	0.5	33.44	5.65	0.98	0.18	-283949.86	-305320.29	-285853.56	-277708.66	-287240.05	-288014.49	9246.53	11
10	0.9	19.46	1.91	0.65	0.03	-304802.35	-310363.78	-297594.00	-289543.72	-298793.34	-300219.44	7025.79	12

Table 6 - K-MEANS DETAILED RESULTS

Clusters	Init	Time	Inertia	Homogeneity	Completeness	V measure	Adjusted Rand Score	Adjusted Mutual Information	Silhouette
14	k-means++	3.611s	584959164	0.006	1	0.012	0	0	0.56
20	k-means++	4.968s	575118026	0.008	1	0.015	0	0	0.707
30	k-means++	11.703s	558941725	0.014	1	0.027	0	0	0.682

Table 7 - NMF DETAILED RESULTS

Clusters (K)	Coherence Score
4	0.2321
6	0.2421
8	0.3054
10	0.3221
12	0.3282
20	0.3607
Parameters of max_iter=2000,init="nndsvd"	

Execution Pipeline

Supervised

Step 1: To parse through the PDFs and create the data frame of lines spoken per character run Scripts_cleaning_process.ipynb inside the **ReadScripts/supervised** folder along with having all the 8flix PDFs of the scripts/teleplays from the folder **Seasons_Episodes** inside the **Data** folder we provided.

- If you would like to run the EDA for the character lines use the file supervised_EDA_final.ipynb along with charlines_df.csv and gender_st.csv inside the **ReadScripts/supervised** folder.
- Feature analysis & Failure analysis (to produce cosine similarity table for character comparison) can be found towards the end of the supervised_tvshow_scripts_logregress.ipynb notebook. The csv creation of the cosine similarity chart found in Table 4 page 19 of the appendix are in the last few cells of the supervised_tvshow_scripts_logregress.ipynb notebook. (All located inside the **ReadScripts/supervised** folder)

Step 2: To run the Logistic Regression portion of supervised learning use the file supervised_tvshow_scripts_logregress.ipynb along with charlines_df.csv and gender_st.csv inside the **ReadScripts/supervised** folder.

Step 3: To run the Random Forest portion of supervised learning use the file supervised_tvshow_scripts_Randomforest.ipynb along with charlines_df.csv and gender_st.csv inside the **ReadScripts/supervised** folder.

- To run the hyperparameter analysis inside the supervised_tvshow_scripts_Randomforest.ipynb notebook at the end please have the Hyper_parameter_tuning_RCF.csv available to make the final bar chart visualization. (All located inside the **ReadScripts/supervised** folder)

Step 4: To run the Linear SVM portion of supervised learning use the file supervised_tvshow_scripts_LinearSVM.ipynb along with charlines_df.csv and gender_st.csv inside the **ReadScripts/supervised** folder.

Step 5: To run the Model Analysis portion of supervised learning use the file Model_Analysis.ipynb along with charlines_df.csv and gender_st.csv inside the **ReadScripts/supervised** folder.

** All supervised learning notebooks contain the basic starting EDA and formation/cleaning steps.

Unsupervised

Step 1: To utilize the PushShift API please use milestone_subreddit_get_subs.ipynb inside the **ReadScripts/unsupervised** folder.

- If you would rather not run this part of the code and skip right to the 3 unsupervised models you would need the output of the notebook above which is the posts_st.txt and located inside the **ReadScripts/unsupervised** folder.
- We also created a sample of 100 rows of data from the posts_st.txt file in csv format if you would rather use named: sample_posts_st.csv

Step 2: To run the LDA portion of unsupervised learning use the file unsupervised_subreddit_full_posts_lda.ipynb along with posts_st.txt (or sample file) located inside the **ReadScripts/unsupervised** folder.

Step 3: To run the K-Means portion of unsupervised learning use the file unsupervised_subreddit_full_posts_kmeans.ipynb along with posts_st.txt (or sample file) located inside the **ReadScripts/unsupervised** folder.

Step 4: To run the NMF portion of unsupervised learning use the file unsupervised_subreddit_full_posts_NMF.ipynb along with posts_st.txt (or sample file) located inside the **ReadScripts/unsupervised** folder.

- In order to compare the models and run the coherence scores you will need our Gensim word2vec saved dictionary, we have included the files needed (w2v-model.bin, w2v-model.bin.syn1neg.npy, w2v-model.bin.wv.vectors.npy) inside the **ReadScripts/unsupervised** folder.
- The unsupervised evaluation section is inside the NMF notebook (unsupervised_subreddit_full_posts_NMF.ipynb) and creates the visuals in the report for sai

** All unsupervised learning notebooks contain the basic starting cleaning, manipulation and process steps to prepare the data for the unsupervised models.