Predicting Movie Genres based on Overview Text Using TF-IDF

By Travis Tran and Lacey Umamoto
December 7, 2020

1. Dataset

Our dataset is made up of movies from the Movie Database (TMDb), themoviedb.org (Rappeneau, “350 000+ Movies from themoviedb.org”). In the original dataset, there are 329,044 rows and 22 columns. The columns included are: id, budget, genres, imdb_id, original_language, original_title, overview, popularity, production_companies, production_countries, release_date, revenue, runtime, spoken_languages, status, tagline, title, vote_average, vote_count, production_companies_number, production_countries_number, and spoken_languages_number.

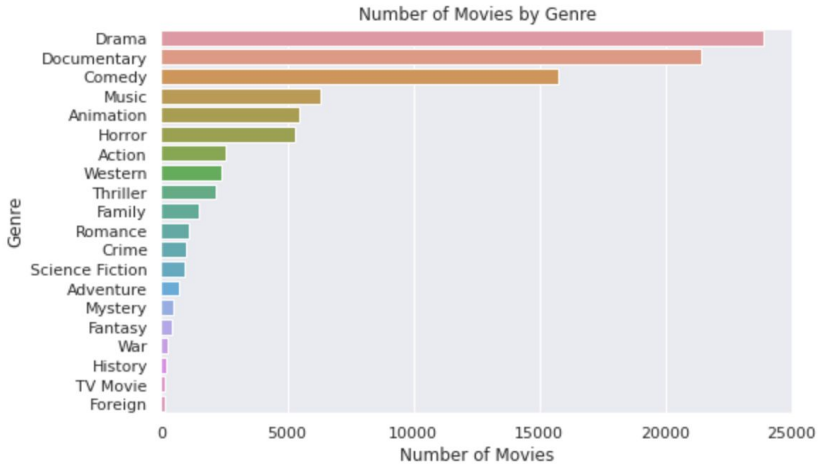
The columns relevant to our project are: id, genres, and overview. To clean our data, we dropped rows with no overviews as well as rows that had "No overview found" or "No overview found." as their overview value. Since we are predicting genres, we dropped rows that had no genres. To simplify our predictive task, we also dropped rows that had multiple genres. We are left with 91,202 movies.

From these movies, their release dates range between December 9, 1874 and December 31, 2020, since there are movies posted on the site that have not yet been released. There are some movies in our dataset that do not have release dates. Out of our 91,202 movies, 2,495 movies (2.7%) do not have release dates.

According to the creator of the dataset, movie databases are centered around white Anglo-Saxon produced movies. Out of all countries in the world, India produces the most movies, but Bollywood movies, for example, are not dominant in the dataset (Wikipedia contributors). Therefore, it should be understood that the dataset is not representative of all movies that have ever been created in the world.

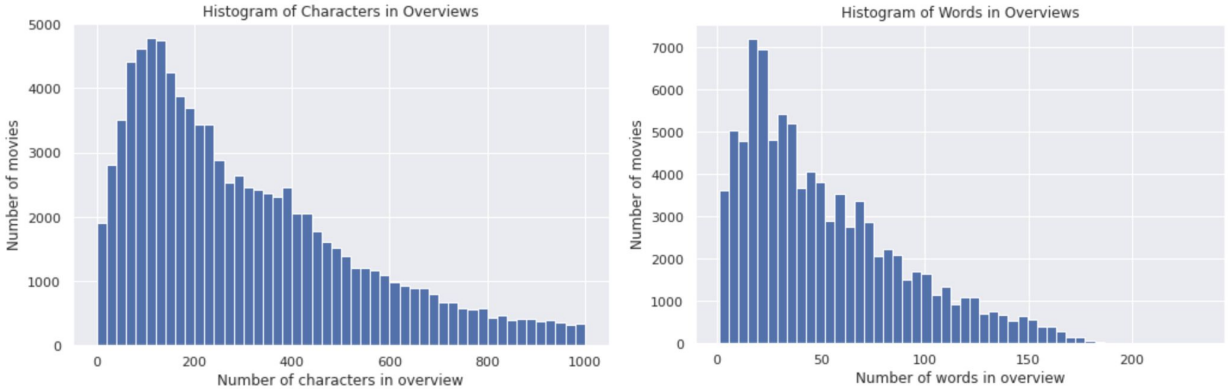
1.1 Genre

There are 20 different types of movie genres: Drama, Documentary, Comedy, Music, Animation, Horror, Action, Western, Thriller, Family, Romance, Crime, Science Fiction, Adventure, Mystery, Fantasy, War, History, TV Movie, Foreign. The top three most common movie genres in our dataset are Drama, Documentary, and Comedy. The movie genres are distributed as follows:



1.2 Overview Length

The first of the two histograms shows the number of characters in overviews over our dataset. It appears that the distribution is positively skewed, and a majority of overviews have fewer characters. The median number of characters in an overview is 247 characters. We choose to consider the median rather than the mean because the graph is skewed and because the median is not affected by outliers. The second histogram shows the number of words in overviews over our dataset. The shape is generally the same as the previous one. We can see a large spike in the graph and can say that a lot of movies like to keep their overviews around 15 to 20 words long. Nonetheless, the median number of words in an overview is 42 words.

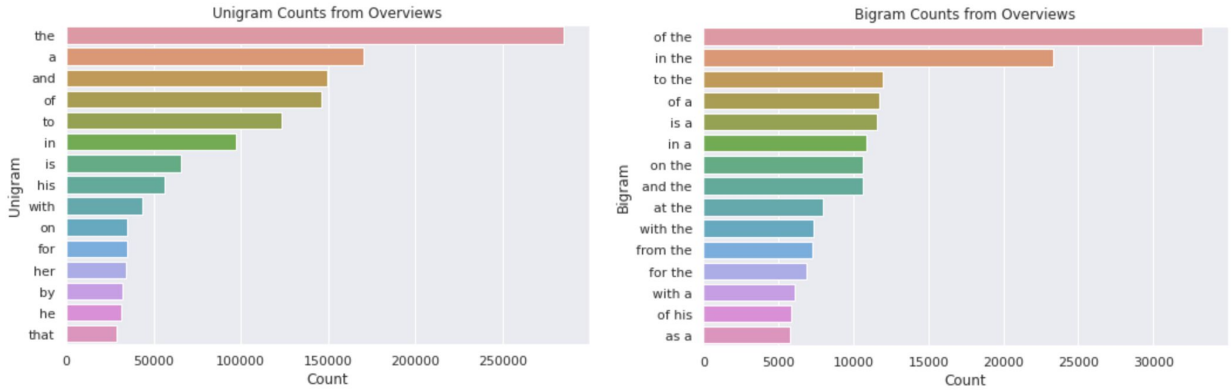


Similar to how we separated the top unigrams and bigrams by genre, we can also break down the number of characters and number of words in overviews by genre to learn if certain genres have distinct median overview lengths. It appears that documentary overviews use the most characters (295) and also the most words (48), whereas animation movies use the least amount of characters (172) and least number of words (30).

genres	# chars median	# words median
Drama	239.0	41.0
Documentary	295.0	48.0
Comedy	232.0	40.0
Music	326.0	55.0
Animation	172.0	30.0
Horror	232.0	40.0
Action	267.0	47.0
Western	247.0	43.0
Thriller	230.0	40.0
Family	253.0	45.0
Romance	235.5	41.0
Crime	178.0	31.0
Science Fiction	240.5	41.0
Adventure	201.5	35.0
Mystery	166.0	29.0
Fantasy	196.0	34.0
War	203.5	35.0
History	205.0	34.5
TV Movie	210.0	36.0
Foreign	258.0	44.0

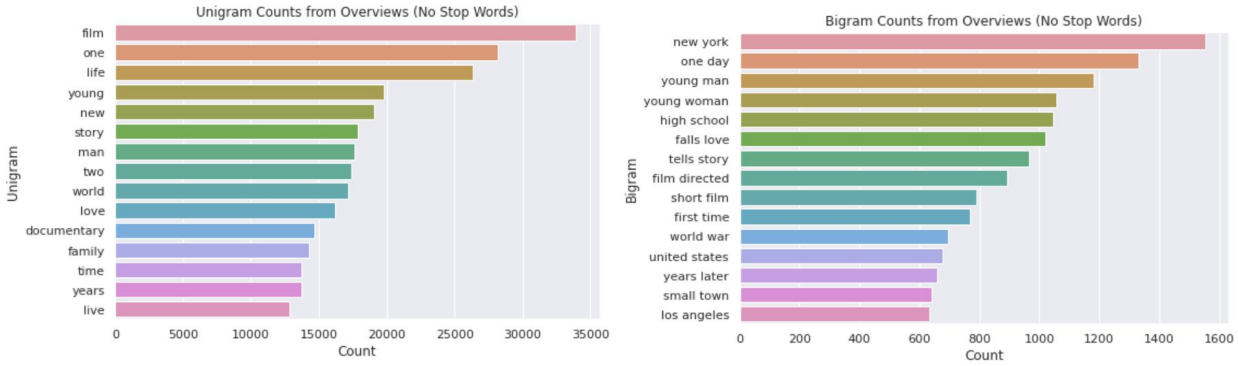
1.3 Unigrams and Bigrams

Further exploring the data, we calculate the top 15 most common unigrams and bigrams from overviews:



Since these words are common in nearly any document of text, these results are expected. We removed stop words (such as "the", "a", "and") from overviews. Our most common bigrams over all genres are simply combinations of stop words. Once we remove stop

words from the overviews, we can form more interesting bigrams. The top 15 unigrams and bigrams are shown below:



This information is more interesting than what was shown in the last two graphs. However, we can take it a step further and separate these counts by genre. As a result, we can see that certain unigrams and bigrams are more prominent within one genre. For example, as shown below, if we ignore stop words, we can see that the top 5 unigrams for the Music genre are live, concert, band, album, and music.

2. Predictive Task

The predictive task that we plan on using for this dataset is a multi-class classification model on the ‘genres’ column. We plan to use TF-IDF for both unigram and bigram to see which one has the best performance. We will evaluate our model’s performance by how accurate each model is compared to the actual data. The baseline model that we will use is the classification of the genre column while using counts of unigram or bigrams as the features.

2.1 Validity and Prediction

The validity of our model predictions will be demonstrated by using a validation set which will find the best parameters for the model then we will predict on the test set. For instance, we will train on the training data, which will be fifty thousand randomly selected rows from the cleaned movies dataframe, afterwards we will predict on our validation data, which will be another fifty thousand randomly selected rows from the cleaned movies dataframe.

Finally, after we get the desired results predicting on the validation data, we will then predict using the testing data, another fifty thousand randomly selected rows, and then that will be our final prediction. This way will show that our model isn’t overfitting to one dataset and will also show that they are significant since we will prove the model is valid since we will have tested on two randomly selected datasets.

2.2 Pre-Processing

For this model to work correctly, we had to pre-process the data to obtain useful features, so the bigram and unigrams we just had to have a count for all the unigrams and

bigrams by going through all the rows, once we got every bigram and unigram count we had to get the top thousand counts and the words, since it would have taken too long for the code to run if we searched for every possible unigram or bigram combination. So for when we make the feature we will only look for the top thousand unigrams or bigrams in the text data, and keep track of their counts for that row then the feature will be a vector of that row's unigram or bigram count.

2.3 EDA Justification

For the TF-IDF feature model, we kept the same top thousand words counts of the unigram and bigram throughout the whole dataset, then we count the number of times the unigram or bigram shows up on the text for that row and the count of rows, then that's all we need for the TF-IDF formula, then you calculate the TF-IDF for each word and that will the model feature of TF-IDFs. From the EDA, we realized this dataset didn't have too many useful columns for predicting the genre of the movie, so we decided to use the 'overview' column which had the best chance of predicting the genres of the movies, so after deciding that the 'overview' column was what we wanted to use, the best approach for textual data was counts of the words or TF-IDF, so that's what we did.

3. Model

3.1 Evaluation

Our best predicting model was using RandomForestClassifier() along with the feature of TF-IDF on the top thousand unigrams. Since the data we decided to work with was textual data, we did not have too many routes to take, so we chose TF-IDF since it works best with textual data. Some of the other models that we considered were linear_model.LogisticRegression(), linear_model.ridge(), and xgb.Booster(). However, none of these models gave a prediction as accurate as the RandomForestClassifier(). These models were not as accurate as RandomForestClassifier() and some of them also took too long to run as well. For the linear_model.ridge(), its pros are that it is really simple, so it is computationally efficient. Since it's so simple, it helps with overfitting. However, the one negative was that it could have some dimensionality reduction. linear_model.LogisticRegression() has many of the same benefits of linear_model.ridge(), but non-linear features do not work too well with logistic regression, since the logistic regression has a linear decision surface, and we are not working with linear data. Finally, we wanted to try xgb.Booster() since it is good for classification and regression, but while testing, it was not more accurate than RandomForestClassifier(). The way we evaluated our model was just by accuracy. The models we chose to use were unigram and bigram count then unigram and bigram TF-IDF, which were both models relevant from classwork.

3.2 Comparison

The counts model did not have as high an accuracy as the TF-IDF model, which is fine since our baseline was the counts model, and we improved on the baseline. The counts model was just a basic count of all the unigrams or bigrams that appeared, which gives each word count the same value. However, for the TF-IDF model, each word has a different value for each time it appears in a document since it is being compared to all the documents in the dataset, so TF-IDF was a bit more effective since it gave each word a different weight depending on how rare or common the word was throughout the dataset. The relevant baselines that we used to be compared were the models that used the unigram and bigram counts as the features.

3.3 Optimization

We optimized our TF-IDF and counts model by getting rid of stop words, since we saw that stop words appeared in a majority of our top thousand unigrams and bigrams, so we filtered out any stop words from the overviews in order to get more value from the other words.

3.4 Issues

We did not have any issues with overfitting since we did not hypertune any features. We only looked at the unigram and bigrams. One of the main issues we had during the beginning of building the models for the baseline and TF-IDF was that it would take too much time to run. We decided to scale down the data to just fifty thousand rows per dataset, and it improved the run time greatly.

We had a little issue with the genre column values where the values would have multiple genres for just one movie, which would mess with our predictions so we got rid of any multi-genre movies in the dataset and changed all the genres to a number so the model could better predict the genre.

4. Literature

We are using an existing dataset from Kaggle created by using the Movie Database (TMDb) API. Ways in which this dataset was used in the past include analyzing the evolution of movies over the years by looking at movies' runtimes, numbers of votes, genres, and the number of movies for a specific genre over time as well as answering the question of whether romance or crime movies sell better (Rappeneau, "Evolution of movies..."; Towhid).

4.1 Similar Datasets

The Movies Dataset on Kaggle includes metadata for 45,000 movies. Some example projects with this dataset include a movie recommendation system, a movie rating predictor, and a movie success predictor (Banik). A more popular dataset is from MovieLens. The next two methods we will look at get their data from MovieLens. One method uses MovieLens 1M, which

has 1 million ratings from 6,000 users on 4,000 movies (“MovieLens”). Each movie from this dataset has an id, title, and genres, but no overview. It also includes a ratings .csv file, where each rating has a user id, movie id, rating, and timestamp.

4.2 State-of-the-Art Methods

Bayesian Approach

Lenskiy and Makita take a Bayesian approach. They use movie ratings as a feature vector. Then, they apply a multivariate Bernoulli event model to calculate the conditional probability of a movie being a certain genre. The model is as follows:

Each movie has a feature vector where each column corresponds to a user u in a set of users U. A cell $v_{u,m}$ is 1 if the user u rated the movie or 0 otherwise. Lenskiy and Makita assume that user ratings are independent. So, if we take the product of the user probabilities of the movie attribute values over all user perceptions of genre information, then we can get the probability of a movie m being a genre g, P(m|g):

$$P(m|g) = \prod_{u \in U} P(u|g)^{v_{u,m}} \left(1 - P(u|g)\right)^{1 - v_{u,m}}$$

where P(u|g) is the user u’s probability of rating a movie of genre g. Each movie is a collection of independent Bernoulli experiments. In other words, there are as many experiments for each movie as there are users in set U. A log transformation is carried out on P(m|g) because with a large number of users, the product would be close to 0. Now, the probability P(u|g), which gives a user’s preferences towards certain genres, is given by:

$$P(u|g) = \frac{1 + \sum_{m \in M} v_{u,m} P(g|m)}{|U| + \sum_{m \in M} P(g|m)}.$$

We finally predict a genre g by selecting the g that has the largest Bayes’ value. Bayes’ theorem is given by:

$$P(g|m) = \frac{P(m|g) \cdot P(g)}{P(m)}$$

where P(g) is the probability that a movie in our dataset is of genre g, or the number of movies of a genre g divided by the total number of movies (Lenskiy and Makita). Lenskiy and Makita used ratings as their features, but since our dataset does not have ratings, we were unable to recreate this method.

Bidirectional LSTM

Ertugrul and Karagoz use bidirectional long short-term memory networks (Bi-LSTM) on movies’ plot summaries. First, they divide plot summaries into sentences and assign the genre

of a movie to each sentence. To represent the summaries, they use pre-trained continuous word vectors of length 300. They were trained using Wikipedia and created using the skip-gram model.

The LSTM model is a recurrent neural networks (RNN) architecture. They use two LSTM networks, thus a Bi-LSTM. One LSTM models the preceding contexts, and the other models the following contexts. To train, each sentence is given as input for the model, represented not only as the words in the sentence but also as its continuous word representations. After being input, they reach the “embedding” layer, where the semantic and syntactic relationships between words are extracted. Then, the word representations reach the Bi-LSTM network, where genres are estimated for each sentence. The “softmax” layer takes in the last output of the Bi-LSTM and finally returns the probabilities of classification for each genre. Using majority voting, the final predicted genre is chosen (Ertugrul and Karagoz). For our features, we used movie overviews, which are similar but still different from plot summaries in that they are much shorter and do not include as much information as the summaries. We did not implement anything we learned from this method.

4.3 Conclusions

The Bayesian approach resulted in a prediction accuracy of 83.8% when they used 80% of their dataset as their training set. When only training with 1% of the training set, they were still able to achieve an accuracy of at over 50% (Lenskiy and Makita).

The Bi-LSTM approach resulted in a prediction accuracy of about 67.7%. This outperformed their baselines of RNN (62.7% accuracy) and linear regression (63.0% accuracy). They also compared separating the summary into sentences versus keeping the summary whole. For Bi-LSTM, separating the summary into sentences resulted in an accuracy increase of 2-3% compared to just keeping the summary whole (Ertugrul and Karagoz).

Out of all of these methods, our model using TF-IDF still performed the best.

5. Results and Conclusions

Describe your **results and conclusions**. How well does your model perform compared to alternatives, and what is the significance of the results? Which feature representations worked well and which do not? What is the interpretation of your model's parameters? Why did the proposed model succeed why others failed (or if it failed, why did it fail)?

Of the different models you considered, which of them worked and which of them did not?
• What is the interpretation of the parameters in your model? Which features ended up being predictive? Can you draw any interesting conclusions from the fitted parameters?

5.1 Performance Comparison

The following is a table showing the prediction accuracies on the test data using the Random Forest Classifier without stop words in our overviews:

	TF-IDF	Unigram	Bigram	Accuracy	n_estimators
0	Yes	Yes	No	0.75066	10
1	Yes	Yes	No	0.76858	30
2	Yes	Yes	No	0.77314	50
3	Yes	Yes	No	0.77272	70
4	Yes	Yes	No	0.77386	90
10	Yes	No	Yes	0.75360	10
11	Yes	No	Yes	0.76580	30
12	Yes	No	Yes	0.77001	50
13	Yes	No	Yes	0.77578	70
14	Yes	No	Yes	0.76891	90

The model that produces the best accuracy uses TF-IDF on the bigrams with 70 estimators, as seen in row 13. It has an accuracy of 77.578%.

5.2 Features and Parameters

Our features were vectors of length 1000 that contained the top 1000 most common unigrams or bigrams. The columns of our feature vectors correspond to a top 1000 word, where a cell in a vector is 1 if the word is present in the overview or 0 otherwise. We tried both keeping in and removing stop words from the overviews. We found that removing the stop words raises our accuracy. Also, using unigrams rather than bigrams resulted in higher accuracies.

In terms of parameters, we decided to manipulate n_estimators, which is the number of trees in our forest for the Random Forest Classifier, and we found that increasing n_estimators raises our accuracy.

5.3 Conclusion

The model that performed the best was the TF-IDF on the bigrams with 70 estimators. It used the Random Forest Classifier and ignored stop words in the overviews. It has an accuracy of 77.578%. For some reason, our bigrams did a bit worse than our unigrams. It might have been because for overviews of movies, adjectives do a better job at distinguishing the difference between the different genres than bigrams do. Also, the counts model performed worse than the TF-IDF model since it did not give the words any weight or value. Counting words like “a”, “and”, and “the” did not help the predictor, likely because those words are commonly used everywhere. WIth TF-IDF, we count the words that hold more weight, which is why TF-IDF performed better.

Works Cited

Banik, Rounak. “The Movies Dataset.” *Kaggle*, 24 Oct. 2017, https://www.kaggle.com/rounakbanik/the-movies-dataset.

Ertugrul, Ali Mert, and Pinar Karagoz. "Movie genre classification from plot summaries using bidirectional LSTM." *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*. IEEE, 2018.

Lenskiy, Artem A., and Eric Makita. "Bayesian approach to users’ perspective on movie genres." *Journal of information and communication convergence engineering* 15.1 (2017): 43-48.

“MovieLens.” GroupLens, grouplens.org/datasets/movielens. Accessed 7 Dec. 2020.

Rappeneau, Stéphane. "350 000+ movies from themoviedb.org." *Kaggle*, 17 Sept. 2017, www.kaggle.com/stephanerappeneau/350-000-movies-from-themoviedborg?select=AllMoviesDetailsCleaned.csv.

Rappeneau, Stéphane. "Evolution of movies number, runtime, number of votes and genre over the years." *Kaggle*, 2017, www.kaggle.com/stephanerappeneau/evolution-of-movies-over-the-years.

Towhid, Nurul Akter. "Romance/Crime." *Kaggle*, 2017, www.kaggle.com/towhid1/romance-crime.

Wikipedia contributors. "Film Industry." *Wikipedia*, 2 Dec. 2020, en.wikipedia.org/wiki/Film_industry.