Movie Review Rating Prediction

Alyssa Liu, Jonah Jung, Joyce Jeon, Memphis Lau, Nathanael Nam, Ryan Kawamura

Abstract:

IMDb, short for the Internet Movie Database, serves as an online repository that provides data and statistics pertaining to movies, TV shows, and other entertainment. We will be analyzing the relationship between user-written reviews to predict the sentiment of imdb reviews. This classification task is accomplished by analyzing the performance of different preprocessing methods (Bag of Words, Word Embeddings) and machine learning algorithms (Logistic Regression, KNN, Decision Tree, and Random Forest).

1. Introduction:

In the year 2023, it is reported that movie theaters in the United States and Canada have sold around 825.2 million tickets. With so many citizens going to the cinema to watch movies every year, there also comes an influx in the number of people wanting to share their criticism and appreciation for films that they have watched. One notable form of feedback comes in the form of online movie reviews. Many movie goers turn to websites such as IMDb and Rotten Tomatoes to post their reviews.

Because anyone is able to share their opinions of movies online, there are various and diverse types of reviews available. This leads to different types of sentence structures, grammar, diction, and modern slang, which can cause confusion in our models. Issues such as numericalization of words and high dimensionality make this research difficult and pose challenges to our analysis.

In this paper, we aim to find the most effective model and natural language processing technique to retrieve the most accurate results for movie review sentiment analysis. We will explore NLP techniques such as Bag of Words, GloVe, and Word2Vec as well as models including Logistic Regression, Decision Trees, Random Forest, and K-Nearest Neighbors to find which technique and model can most accurately predict the positive or negative sentiment of movie reviews. Being able to train a model to do so could be very beneficial to the film industry in ways such as helping filmmakers understand audience's feelings towards movies, helping film marketing teams direct their strategies towards their target audience, and aiding platforms in aggregating review scores more accurately.

2. Dataset Description

IMDb is a comprehensive online database that serves as a resource for information about films, television series, podcasts, and other sources of internet entertainment. It provides an extensive collection of data, including details about movies, cast and crew information, ratings, and reviews.

Its extensive repository of reviews contains a wide variety of opinions, sentiments, and critiques, making it an ideal source to conduct sentiment analysis.

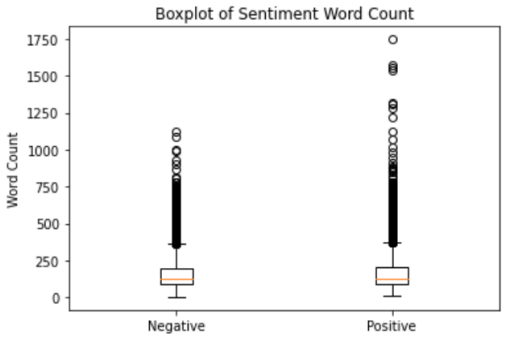

Figure 1: Box Plot of Word Count for Positive and Negative movie reviews

The IMDb dataset consists of 50,000 movie reviews, where each review is assigned a binary sentiment of either positive or negative. Our goal is to leverage different preprocessing methods and classification models to best predict the sentiment of IMDb movie reviews.

3. Data Preprocessing:

Before implementing any classification modeling, it is necessary to clean and preprocess the data to transform the raw text into a more structured and standardized format. This will help eliminate noise and enhance the model's efficiency and accuracy by only focusing on components that are associated with sentiment. We also convert the labels to integers, with "positive" corresponding to 1 and "negative" corresponding to 0.

We first began by removing punctuation and HTML tags in the review text, as this is typically irrelevant for text analysis. Next, we normalize the data to make the text data more consistent so that the model focuses on patterns in the data that contribute to sentiment rather than extra noise and miniscule case variations. We do this by converting all letters to lowercase and removing multiple consecutive spaces to further standardize the review text [2]. Then, we remove less informative words including personal pronouns, determiners, coordinating conjunctions, and prepositions so that we can focus on more content-rich words in the text [3].

Finally, we turn to the task of lemmatization. Lemmatization consists of reducing words to their root forms, but it ensures that these words are still meaningful representations. This process is more sophisticated and time-consuming than stemming for example, as stemming simply removes prefixes and suffixes of words. We choose lemmatization because it takes into account each word's part of speech to produce accurate and meaningful results [1]. During this process, we used WordNet, a lexical database, to check each word in the processed text to identify its part of speech. We then lemmatize each word using the WordNet lemmatizer, which considers the part of speech to accurately reduce each word to its root form.

After preprocessing the dataset, we created word clouds for both the negative and positive reviews to identify common keywords in each. Though there are many expected similarities between the two like "movie" or "film", we see some clear differences like "well", "see movie", and "love" in the positive reviews and "nothing", "suppose", and "bad film" in the negative reviews.

Before proceeding, we split the dataset into training and testing sets, each with 25,000 reviews and labels. Because there are already significant text differences between the positive and negative reviews, we do not proceed with adding additional features. We explored review lengths as shown in Figure 1, but aside from some positive reviews having long length, both positive and negative reviews appear to generally have the same length.

Since machine learning algorithms are not capable of interpreting raw text directly, we turn to feature extraction to convert the text into a numerical representation. In this case, we used three different preprocessing methods: Bag of Words and Word Embeddings [4].

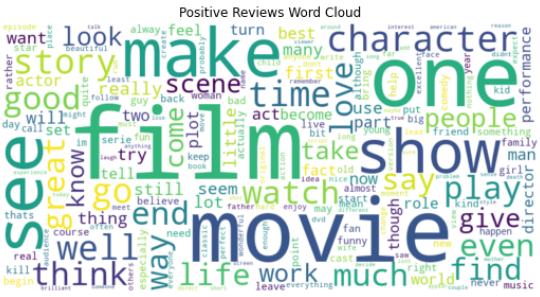

Figure 2: Positive/Negative Reviews Word Clouds

3.1 Bag of Words

Bag of Words was one of the approaches used to turn the review text into numeric representation. We specifically used sklearn's CountVectorizer(), which turns text into numeric vectors. This method is first fit to the training data, and transforms it by counting the number of occurrences of each word in the training set. Each word is given a unique representation [5].

When using the default CountVectorizer(), the resulting shape of the training data was (25000, 111931), indicating there were 111,931 unique words in the training data. The method was then used to transform the testing data, yielding a shape of (25000, 111931). To ensure consistency between the number of unique words between the training and testing data, words that appear only in the testing data are ignored.

While experimenting with hyperparameters, we found that providing the argument ngram_range = (1, 2) generally yielded a higher accuracy for all the models considered, even though it took a bit longer to run. This means that countvectorizer considers both single words (unigrams) and unique pairs of consecutive words (bigrams). After using this transformation, both the training and testing data had dimensions of (25000, 1643870), indicating that there were 1,643,870 unique unigrams and bigrams found in the training set altogether.

3.2 Word Embeddings: Word2Vec/GloVe

Word Embeddings are a popular technique in natural language processing that attempts to represent words as vectors in a continuous vector space. This method captures the semantic relationships between words. For example, the vector representation of "king" added to the vector representation of "female" may produce the word "queen". For our project, we use pre trained embeddings that already numerically represent most words in a dictionary. We use Google's Word2Vec API, which converts words to a 300-dimensional vector, as well as glove-wiki-gigaword-50, which represents words in a 50-dimensional space. In order to apply word embeddings to an entire text, we transform texts by finding the 50 or 300-dimensional vector for each word in a text, then computing the average vector across all those words. Our theory is that this average captures all the important information from the words in the text [7]. Thus, for Word2Vec, the dimension of X_train is (25,000, 300), and for GloVe, the dimension is (25,000, 50) [6].

4. Modeling

4.1 Logistic Regression

Logistic Regression is a parametric model typically used for binary classification purposes. It estimates the conditional probability $P(Y = 1 \mid X)$, and assumes a linear relationship between the independent variables and the log odds of the dependent variable. The model uses the sigmoid function, which takes input and maps it to the probability of the data belonging to one of the two categories [7].

The combination of the bag of words model considering both unigrams and bigrams, and logistic regression yielded an accuracy of 0.8798. The transformed training and testing datasets both had lengths of 1,643,870, which is the number of unigrams and bigrams that were counted. This was the most successful out of all the NLP approaches.

To find a more optimal Logistic Regression model for this dataset, we used *GridSearchCV* (grid search cross-validation) to experiment with hyperparameter combinations for CountVectorizer() and LogisticRegression(). We did the same for both versions of Word Embeddings, but the bag of words approach still yielded the highest accuracy. For the count vectorizer, we primarily experimented with ngram_range, which allows either single words, bigrams, or trigrams to be considered to possibly capture more context. For logistic regression, we looked at C, which

is the inverse of regularization strength and a penalty of either L2 regularization or no regularization. Each combination of parameters was run with 3-fold cross-validation [8]. The highest accuracy yielded with this technique was 0.89892 with parameters C = 1, L2 regularization, and ngram_range = (1, 2), which was very promising. We proceed by experimenting with more complex models to try and increase the accuracy score.

4.2 KNN

K-Nearest Neighbors (KNN) is a simple machine learning algorithm that can be applied to classify movie reviews as either positive or negative, basing the classification on the majority class of its K-Nearest Neighbors in the feature space. This can be applied to the dataset after preprocessing and is good because it is a very simple model to understand and does not assume a specific form for the underlying data distribution. KNN is a nonparametric model [9].

Because KNN is a distance-based algorithm, we typically must scale the input data to ensure everything is on the same scale. However, in the case of the bag of words approach, the features are already on the same scale because each number represents frequency counts. For word embeddings, we applied standard scaler normalization.

When proceeding with KNN, we used cosine similarity for the Bag of Words approach specifically because each vector is already of the same length. This inherently captures the relative proportion of features in the document, as it measures the cosine of the angle between vectors rather than the distance between two points.

We performed hyperparameter tuning for the KNN classifier combined with the word embeddings approach, as this NLP technique was the most successful with this model. We proceeded with the Euclidean distance metric. This combination obtained the highest accuracy to start with. We further tuned the parameters of KNN with *GridSearchCV*. We only included odd numbers for k, to ensure that there wouldn't be ties between the two classes, and then we proceeded with the parameters that yielded the highest accuracy score. The k-Nearest Neighbor classifier configured with k=23 neighbors demonstrated the best performance for the model. Our optimal KNN model obtained an accuracy of 0.7924 with the word embeddings (word2Vec) approach.

4.3 Decision Tree

The Decision Tree algorithm classifies movie reviews as positive or negative by partitioning the dataset, creating a predictive tree for sentiment analysis.

After preprocessing, each word in a review becomes a feature. The Decision Tree algorithm identifies the most predictive words by selecting features that best split the data into subsets reflecting different sentiments. The tree grows by iteratively choosing optimal features until a stopping criterion is met, which, in our case, was the maximum tree depth. At this point, the algorithm assigns a 'positive' or 'negative' sentiment to leaf nodes based on the majority proportion of samples in the leaf. The leaf's label is the predicted sentiment for the review [10].

To maintain consistency with the other models tested in this research, the training and testing datasets for the decision tree algorithm were also tested on 1,643,870 unigrams and bigrams, which were extracted from the 25,000 training reviews in the IMDb dataset.

We initially started by implementing a standard Decision Tree model with no adjustments to the hyperparameters of the model. This yielded a low accuracy of 0.72. To improve the performance of the model, we used GridSearchCV on all three preprocessing methods to find the optimal combination of key hyperparameters: the minimum number of samples required to split a node, the minimum number of samples required to be at a leaf node, and the maximum number of features to

include in the model. Determining the most optimal values for these key parameters can help prevent overfitting and optimize the performance of the decision tree. Looking at the performance for both text preprocessing methods, Bag of Words had the highest performance.

Using the Decision Tree algorithm, coupled with the Bag of Words preprocessing method, we were able to conclude that the optimal configuration of the model consisted of a maximum tree depth of 16, a minimum of 10 samples for splitting, a minimum of 1 sample in a leaf node, and consideration of all features for the best split. These refinements improved the model's accuracy score to 0.7458. Despite the efforts to improve the performance by tuning the hyperparameters, the accuracy only increased to 0.7458. Recognizing the limitations to the Decision Tree model, alternative models, such as Random Forest, may offer better performance as random forest leverages multiple decision trees to yield higher accuracy.

4.4 Random Forest

Following the results of the Decision Tree algorithm, we decided to implement a Random Forest algorithm to see if it would better optimize the accuracy.

Random Forest is a powerful machine learning algorithm that combines a large number of individual decision trees to improve the overall results of classification and regression tasks. Random Forest generally performs better than Decision Tree because of its implementation of randomness in the sampling of the data (bootstrap sampling) and the randomness in the subset of features considered for splitting at each node in a decision tree. This stochasticity allows for a more robust and higher accuracy model. In the case of analyzing movie reviews and performing sentiment analysis, we utilized the Random Forest to classify the movie reviews into positive and negative classifications.

When testing the Random Forest model, we initially started with a base model with 100 estimators. This combined with the previously described Bag of Words approach yielded a score of 0.8488, which was the highest of all the NLP techniques. After seeing the relatively high accuracy of this model, we continued to tune the parameters. Utilizing the GridSearchCV function in python, we were able to test multiple parameters and find the most optimal parameters for this dataset. In Random Forest, there is a hyperparameter *n_estimators*. This argument specifies the number of trees in the random forest. The more trees in the forest, the more robust the model becomes, making it better at generalizing the data. However, too many trees can increase computational complexity without significant gains in accuracy. We tested the model when *n_estimators = 50, 100, 300, 500, 700.* As a result, the best Random Forest model was when *n_estimators = 700*. This model resulted in an accuracy score of 0.8686.

The more trees that are in a Random Forest usually translates to more accuracy. However, the model converged to an accuracy score of 0.869. Thus, we decided that the very minimal increase in accuracy from increasing the number of trees was not worth the computational expense. Thus, we kept the accuracy at 0.8686 with *n_estimators = 700.* This was an improvement over the previous decision tree model. However, it was still not as high as our logistic regression model.

5. Results

To analyze the performance of each preprocessing method and each machine learning algorithm, we compared each of their accuracy and F1 scores. For each of the results, we ran GridSearchCV to tune the hyperparameters and ensure maximum performance.

5.1 Bag of Words

	Accuracy	F1-Score
Logistic	0.89892	0.9071
KNN	0.6636	0.6566
Decision Tree	0.7458	0.7442
Random Forest	0.8686	0.8696

5.2 Word Embeddings: Word2Vec

	Accuracy	F1-Score
Logistic	0.858	0.8585
KNN	0.7924	0.7815
Decision Tree	0.709	0.715
Random Forest	0.776	0.775

5.3 Word Embeddings: GloVe

	Accuracy	F1-Score
Logistic	0.7624	0.76
KNN	0.714	0.703
Decision Tree	0.657	0.654
Random Forest	0.723	0.723

6. Conclusion

6.1 Analysis of Results

Our comprehensive analysis of various natural language processing techniques and machine learning models has yielded insightful conclusions regarding the effectiveness in sentiment analysis of movie reviews.

We first observed that the GloVe and Word2Vec language processing techniques yielded worse performances on the dataset than Bag of Words for many of the selected models. This poor performance is due to the dimensional reduction that takes place in these language processing techniques. GloVe reduces the text to a 50-dimensional vector and Word2Vec performs dimension reduction, both which cause the text to lose too much information, which leads to less accurate results. Although these techniques are computationally less expensive, the decrease in accuracy that occurs as a result does not support the use of these language processing techniques. Thus, Bag of Words was much more effective.

In terms of the machine learning models, we found a distinct hierarchy in the performance with our high-dimensional data: Logistic Regression was the most effective, then Random Forest, Decision Tree, and K-Nearest Neighbors. This trend can be attributed to the inherent characteristics of the models. Logistic Regression is particularly adept at working with high-dimensional, sparse data, which is a common feature in textual data. Random Forest, although effective, struggled with the curse of dimensionality that was present in the data along with the sparsity of the data, which led to a less effective model than our Logistic Regression model. Decision Trees, making up the estimators of the Random Forest, share the same limitations as Random Forests, while also being more prone to overfitting, especially in high dimensional spaces, which can be observed by its less accurate model. Finally, due to KNN's reliability on distance metrics in the model, the model falls short in high-dimensional spaces where distance is less effective. However, this situation shifts when in the cases of GloVe and Word2Vec language processing because of the reduction in dimensionality that takes place.

The reduced dimensions mitigates the curse of dimensionality that occurred in KNNs, making it more effective than Decision Trees in these cases.

Along with finding that Logistic Regression with Bag of Words language processing yielded the most accurate results, our research also highlights the fact that the best model depends on various factors, such as the features of the data, the task at hand, and the preprocessing technique.

Our final model is the combination of the bag of words approach, specifically Countvectorizer(ngram_range = (1, 2)) combined with LogisticRegression(C = 1, penalty = 'l2').

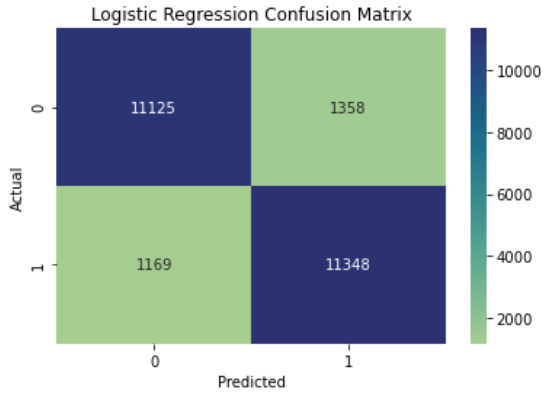

Figure 3: Confusion Matrix of Final Model

6.2 Limitations and Future Work

As mentioned before, we attempted using PCA for dimension reduction in our Bag of Words model. Since the model had over a million dimensions, we thought it would be wise to reduce these into a smaller dimension space. However our attempts failed. PCA crashed our code on each attempt and this might have been due to a lack of computational power and a lack of a GPU present during modeling. For this reason, we decided to skip PCA, however in the future it is definitely worthwhile to examine PCA.

We also experimented with using the downloaded lexicons provided on Kaggle [11], which include a list of positive-associated words and negative-associated words. We used this by filtering out every word in every text that is not present in either of these lists. This process took 24 minutes even while connected to a GPU. At the end of this pre-processing, every text is reduced to only "positive" or "negative" words. Running the bag-of-words technique on this new dataset and then applying logistic regression led to similar results as when we didn't use the downloaded lexicons. We deduced that the results of using the lexicons were not worth the computational resources needed to filter every text in this way.

6.3 Conclusion

We found that the machine learning algorithms outputted different levels of accuracy depending on the preprocessing method that was used. It is important to assess the validity of these models in the context of working with high-dimensional data. Out of all the machine learning models, the Logistic Regression seemed to work the best. This may have been because Logistic Regression is a linear model, which tends to work well in high dimensional spaces since it can handle a large number of features without a significant loss of performance. Additionally, since logistic regression is a relatively low-complexity model, it is less prone to overfitting. The tuning of logistic regression was also quite efficient compared to the tuning of other models.

The next model that worked the best was the Random Forest model, which can accurately identify important features of data because it can handle complex relationships. In the context of sentiment analysis, it can capture the nuanced patterns in the IMBd text data and can distinguish between positive and negative sentiment.

The next model that did not run as efficiently was the Decision Tree model. This may have been because Decision Trees are typically sensitive to noise and can overfit on high-dimensional data.

However, the worst model we ran was the KNN, which outputted the lowest accuracy score. The KNN model does not work well with high dimensional data because as the number of dimensions increases, the distances between the instances become less meaningful, so the effectiveness of the nearest neighbors diminishes. Additionally, we find that the model is computationally expensive, as the model requires calculating the distances between instances.

When you observe lower dimensions, you find that KNN works better than a Decision Tree for word embeddings. This is because KNNs are strong at capturing local structures in lower dimensional spaces where the dimensions have more meaningful interpretations. Also, the simplicity and local emphasis of KNN outperform decision trees that have the potential to overfit in higher dimensions.

A majority of the project placed a large emphasis on preprocessing methods of the data, which was categorized into Bag of Words and Word Embeddings.

The CountVectorizer from scikit-learn was used to implement Bag of Words. The resulting representation had dimensions of (25000, 1643870) for the training data, indicating 1,643,870 unique unigrams (single words) and pairs of words (bigrams). The testing data was transformed to match this vocabulary. The preprocessor then counts the occurrences of each word in the training set, creating a sparse matrix where each word is represented by a unique index. This method disregards the order of words but captures the overall frequency of each term in the dataset.

Next, Word2Vec and GloVe were used to obtain pre-trained embeddings. The dimensions of the resulting representations were (25000, 300) for Word2Vec and (25000, 50) for GloVe. In practice, word embeddings represent words as continuous vectors in a multi-dimensional space, capturing semantic relationships. The embeddings are applied to entire texts by averaging the vectors for each word. This method aims to preserve the semantic meaning of words and their context within the text. GloVe performs worse than Word2Vec because condensing the text to a 50-dimensional vector loses too much information.Word2Vec performs worse than Bag of Words because of the reduction of dimensions. While it does save computation time, the dataset is not too large, and we have the computational resources to withhold the large dimensions of Bag of Words.

Contribution Statement

All group members contributed equally to this project both in writing and in code.

References

[1] Saumyab271. (2022, June 28). Stemming Vs Lemmatization in NLP: Must-Know Differences. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2022/06/stemming-vs-lemmatization-in-nlp-must-know-differences/#h2_4

[2] Amponsah, Josephine. “Preprocessing Text Data for NLP Models.” Medium, 13 June 2023, josephine-amponsah.medium.com/preprocessing-text-data-for-nlp-tasks-57225264762a.

[3] Duong, Huu-Thanh, and Tram-Anh Nguyen-Thi. “A Review: Preprocessing Techniques and Data Augmentation for Sentiment Analysis - Computational Social Networks.” SpringerOpen, Springer International Publishing, 6 Jan. 2021, computationalsocialnetworks.springeropen.com/articles/10.1186/s40649-020-00080-x.

[4] Bordoloi, Monali, and Saroj Kumar Biswas. “Sentiment Analysis: A Survey on Design Framework, Applications and Future Scopes.” Artificial Intelligence Review, U.S. National Library of Medicine, 20 Mar. 2023, www.ncbi.nlm.nih.gov/pmc/articles/PMC10026245/.

[5] Jason Brownlee. (2019, March 12). A Gentle Introduction to the Bag-of-Words Model. Machine Learning Mastery. https://machinelearningmastery.com/gentle-introduction-bag-words-model/

[6] Korab, Petr. “The Most Favorable Pre-Trained Sentiment Classifiers in Python.” Medium, Towards Data Science, 12 Sept. 2023, towardsdatascience.com/the-most-favorable-pre-trained-sentiment-classifiers-in-python-9107c06442c6.

[7] Saini, A. (2021, August 3). Logistic Regression | What is Logistic Regression and Why do we need it? Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/

[8] Mashalkar, Atharva. “Sentiment Analysis Using Logistic Regression and Naive Bayes.” Medium, Towards Data Science, 7 Mar. 2021, towardsdatascience.com/sentiment-analysis-using-logistic-regression-and-naive-bayes-16b806eb4c4b.

[9] Srivastava, Tavish. “A Complete Guide to K-Nearest Neighbors (Updated 2023).” Analytics Vidhya, 19 Oct. 2023, www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/.

[10] Saini, A. (2023, September 13). Decision tree algorithm - A complete guide. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/

[11] Data, NLTK. “Opinion Lexicon.” Kaggle, 21 Aug. 2017, www.kaggle.com/datasets/nltkdata/opinion-lexicon. Accessed 08 Dec. 2023.