Sentiment Analysis of Movie Reviews with Machine Learning and Deep Learning Methods

Jerry Yang, Leow Yi Ling Felise, Ngu Jiahao, Tay Kaiying Roydon

Introduction

In this project, we experiment with Machine Learning (ML) and Deep Learning (DL) techniques to conduct sentiment analysis (SA) of movie reviews. SA involves analysing text data to determine whether the sentiment towards a movie is positive or negative. This is important as it can provide valuable insights into audience opinions and preferences, helping filmmakers and studios understand current trends and interests of moviegoers, and thus make informed decisions about what movies to create.

For a similar problem of movie review sentiment analysis, studies have achieved moderate success using ML algorithms such as Random Forests and Multinomial Naïve Bayes for the task of sentiment analysis. One study achieved an accuracy of 88.50% with a Multinomial Naïve Bayes model (Rahman and Hossen, 2019), while another study achieved 88.95% with its Random Forest classifier (Sahu and Ahuja, 2016). A study has shown the effectiveness of using the Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer algorithm to create text embeddings that can be used for ML classification problems (Yoo and Yang, 2015). Hence, we decided to experiment with the TF-IDF vectorizer and the Count vectorizer algorithm, another frequency-based text embedding algorithm for feature extraction. Transformer models have achieved ground-breaking success in NLP tasks by using the attention mechanism (Vaswani et al., 2017). Building on that, a study achieved high performance on benchmarking NLP tasks with their proposed Bidirectional Encoder Representations from Transformers (BERT) language representation model (Devlin et al., 2018). By leveraging BERT's pre-trained representations of text, neural networks can be fine-tuned for specific tasks, achieving state-of-the-art performance across various NLP applications (Sun et al., 2019). However, past studies used the Adam optimiser when fine-tuning BERT models, which was found to be outperformed by AdamW, which can generalize the model better (Loshchilov & Hutter, 2019). The related studies also did not share methods of hyperparameter tuning for ML and DL, as well as how one should deal with datasets with class imbalance. Furthermore, they did not explore the impact of sentiment class variations. We improved upon these in our project.

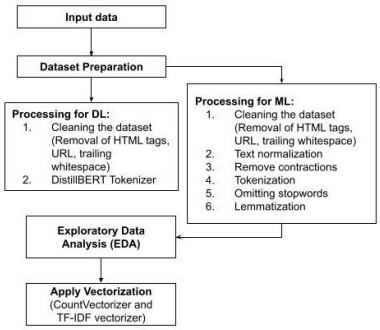

Figure 1: Steps taken to process data.

Dataset

Before experimenting with ML and DL methods, we did some data processing to remove or reduce the noise and variability in the text data and make it more uniform and structured. This can help models focus on the meaningful and relevant information in the text and improve their efficiency and effectiveness. While ML methods require more processing of data, the DL approach using BERT models does not require some processing steps such as dropping stop words and punctuations, since it uses these as context to understand text as well.

Processing for):

1) **Cleaning the dataset:** Remove irrelevant data like HTML tags, URLs, emojis, and punctuation using regular expressions (RegEx) package.
2) **Text normalization:** Convert all text to lowercase for consistency. Lowercasing ensures consistent treatment of words (e.g., "Hello" to "hello"), aiding in text comparison and processing. It also normalizes text, reducing complexity and unique words.
3) **Handling contractions:** Expand contractions (e.g., "don't" to "do not") and remove punctuation. Doing so improves text clarity and removes ambiguity while removing punctuation helps standardize text.
4) **Tokenization:** Splitting text into tokens that can be processed by text embedding algorithms (TF-IDF, BoW and BERT)

5) **Omitting Stopwords:** Remove stopwords (e.g., “I”, “it”, “you”) as they have no meaning in sentiment analysis.
6) **Lemmatization:** Reduce words to their base form (e.g., "running" to "run"), improving text analysis accuracy and model performance.

Processing for Deep Learning (DL):

1) **Cleaning the dataset:** Remove irrelevant data such as HTML tags and URLs.
2) **DistillBert Tokenizer:** Does Wordpiece tokenisation on input text before converting tokens into token IDs that map to the model’s vocabulary, adds special tokens for sentence boundaries and ensures equal sequence lengths through padding or truncation.

Exploratory Data Analysis (EDA)

To analyse both the dataset with positive sentiments and negative sentiments, various EDA are conducted. Before conducting our N-gram analysis, we utilized a boxplot to identify and remove outliers in terms of word length, which can help reduce noise and enhance the signal-to-noise ratio in the dataset, to improve the accuracy of our machine learning (ML) and deep learning (DL) analyses.

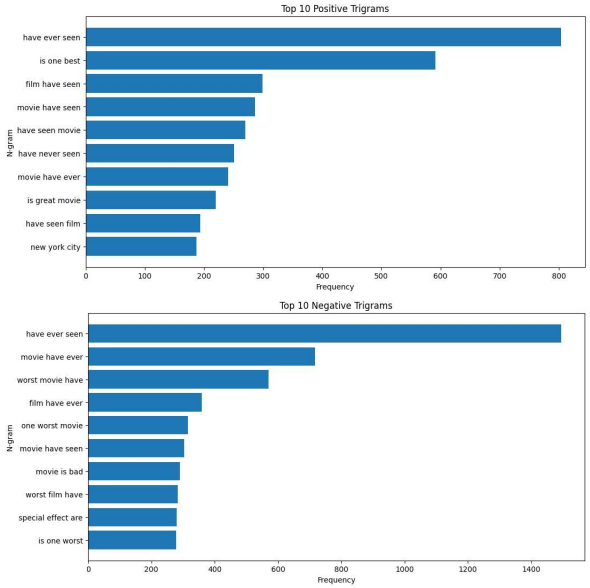

Figure 2: Top 10 Positive (top) and Negative (bottom) Trigrams

From our trigrams analysis, both positive and negative datasets share some of the most common trigrams, such as "have ever seen" and “movie have ever”. This may be because reviewers compare movies, they make comparisons to other movies they have watched. This may be problematic during model training since some of the highest frequency trigrams do not differentiate between positive and negative reviews. However, other trigrams included words like "worst" or "bad" in negative reviews, and “great” or “best” in good reviews, which could be informative to models. This is likely because reviewers use strong language to express their dislike clearly and emphatically, leading to a higher frequency of such words in negative reviews. The trigram "New York City" was among the top positive trigrams but does not contain information about the sentiment of the review. Noise in the data like this may cause the model to learn spurious patterns and false correlations, leading to poorer prediction accuracies.

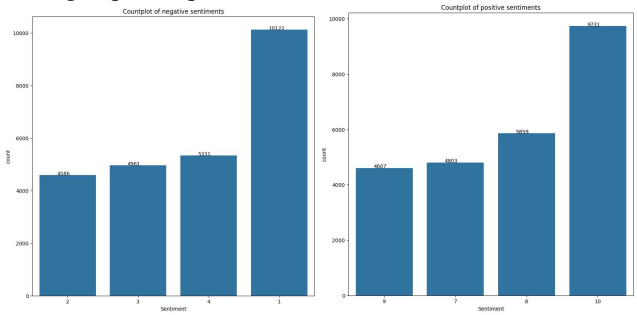

Figure 3: Countplots of ratings. Negative (left), Positive (right)

From Fig 3, we see that viewers are more likely to give extreme ratings (1 or 10). This causes the rating classes to be imbalanced in the dataset.

Methods

The imbalance in the target variable can lead to the training of biased models since some classes may be under-represented in the training data. We conducted random oversampling to improve on this limitation of related works. Random oversampling works by replicating randomly selected instances of the minority classes. This gives the model a more balanced dataset to train with and make better predictions for the minority class.

1. Machine Learning Models

Before utilising the ML algorithms, we used the Count Vectorizer (to create the BoW representation of the text) and TF-IDF vectorizer to transform the training and test data into embeddings. The Count Vectorizer transforms the text data into matrices representing the frequency of each word. The TF-IDF vectorizer transforms a collection of text documents into numerical feature vectors, where each feature represents the importance of a term within a document relative to a collection of documents.TF-IDF of a word is calculated in 2 parts: First, calculate the term frequency (TF). This is the number of times a word appears in the document divided by the total number of words in that document. Inverse Document Frequency (IDF) calculates the importance of this word across all documents. It is given by the formula: log (no. of documents/no. of documents containing the word). The ratio (no. of documents/no. of documents containing the

word) is inverted to give a higher value to words that are less common among all the documents (Maneja, 2021). Both methods help transform text into a form machine learning models can use as input, effectively extracting features from textual data. We used ngram_range = (1,3), min_df = 2, and max_df = 0.85 values for the TFIDF vectorizer. We set the ngram_range parameter to include unigrams to trigrams. N-gram representations capture both individual words and their context, enriching their interpretative power. The min_df parameter controls the filtering of rare n-grams and reduces noise, while the max_df parameter controls removal of overly common terms, focusing the model on meaningful, discriminative features. These settings collectively enhance the model's ability to generalise and make meaningful predictions from textual data. Since related studies have used Random Forest Classifier and the Multinomial Naïve Bayes model, we decided to test these two models, to see which performed better for our project. We also tested the Logistic Regression model since it is a common baseline model for binary classification tasks.

A) Random Forest Classifier

Random Forest Classifier is an ensemble learning method, which works by constructing multiple decision trees during training and outputs the mode of the individual trees. This approach helps to reduce overfitting and increase the overall accuracy of the model. Bagging is a technique used in Random Forest where each tree is trained on a bootstrapped subset of the original dataset. This means that each tree is trained on a different subset of the data, which helps to introduce diversity into the ensemble. By combining the predictions of these individual trees, Random Forest can make more accurate predictions than any single decision tree, reducing overfitting.

B) Multinomial Naive Bayes

Naive Bayes (NB) is a classification algorithm that assumes features are independent of each other. It's not a standalone algorithm but a group of classification algorithms. The math behind NB is based on Bayes' theorem, calculating the probability of a class given a set of features as follows: $P(x|Y) = P(Y|x) * P(x) / P(Y)$
In this context, X represents the class variable, while Y is a dependent feature vector. P(x) and P(x|Y) refer to the priori and posteriori probabilities of x and Y respectively. Multinomial Naive Bayes is ideal for discrete frequency count features, like word counts.

We did hyperparameter tuning on all three models, using both GridSearchCV and RandomizedSearchCV from the scikit-learn library for each. With a given parameter grid, GridSearchCV tries all possible combinations of hyperparameters, evaluates it against a metric and returns the hyperparameters of the model that optimises the specified metric. We used the accuracy metric. RandomizedSearchCV works in the same way as GridSearchCV, the only difference is it randomly selects combinations of hyperparameters instead of trying all combinations from the search space. For GridSearchCV and RandomizedSearchCV, we did a 3-fold cross-validation of training data for each set of parameters, to ensure the hyperparameter tuning does not overfit the models. For Multinomial Naive Bayes, we see an improvement in accuracy from 85.5% to 89.2% after tuning. For Random Forest Classifier, we see an improvement in accuracy from 86.0% to 86.3% after tuning. We also managed to reduce overfitting of the model as the discrepancy between train accuracy and test accuracy has decreased. For Logistic Regression, we got an improvement in accuracy from 89.1% to 90.6% after tuning.

2. Deep Learning Model

We also experimented with the finetuning of transformer models. The finetuning process we adopted involved training a classification neural network that makes predictions using the embedding outputs of a pre-trained BERT encoder model. There are several advantages of using BERT embeddings. Firstly, BERT was trained over a large corpus of data. Secondly, BERT was trained to have a bidirectional understanding of text, through unsupervised learning of predicting words in sentences masked by a masked language model. BERT was also trained to understand relationships between sentences through unsupervised learning of next-sentence prediction. Finally, BERT is a transformer model which leverages the attention mechanism. It has trained 'weight' matrices that help the encoder model create contextual embeddings based on its understanding of how words relate to the rest in the sequence. These factors allow BERT to understand the nuances of language and generate embeddings with representative features useful for downstream classification

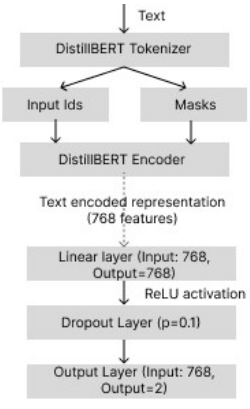

Figure 4: Flow Chart of Neural Network Architecture

tasks. We also experimented with DistillBERT. It was created through knowledge distillation of the BERT model, which is the process of transferring knowledge from a large model to a smaller model that is less computationally expensive to run, without significant loss in performance. We decided to use DistillBERT since doing so allowed us to run more epochs to train our classification neural network with our computation resources, achieving better accuracy as compared to our experiments with the BERT-base. From Fig 4, the DistillBERT Tokenizer does Wordpiece tokenization on text and returns matrices of token IDs representing the wordpieces and attention masks, which tells the encoder model whether the token is a padding token; both are required as model inputs. The DistillBERT encoder model takes these inputs and generates contextualised hidden state vector embeddings with 768 features using the attention mechanism. Embeddings are passed to a fully connected linear layer with the ReLU activation function. Adding dropout layers helps prevent overfitting in neural networks by randomly deactivating some neurons during training, improving generalization to unseen data (Srivastava et al., 2014). We hence added a dropout layer after the linear layer, before generating output from the final layer of 2 output nodes. The class predicted is based on the node with the greater output value. An optimiser is used to update the model's weights during training, while a loss function is used to calculate the difference between the model's prediction and the actual value. We used the AdamW optimiser as an improvement from past studies, and the Cross Entropy loss function for model training, following related studies. Bayesian Optimization was used for hyperparameter tuning of the learning rate and dropout rate. It builds a probability model of the objective function (accuracy with respect to hyperparameter values) and uses it to search for optimum parameters. Due to the large dataset and long training time, we conducted hyperparameter tuning over a subset of the train data (500 train and 50 evaluation rows). We found that a learning rate of 1e-5 and dropout value p=0.1 gave the highest performance.

Results & Discussions

We used the following metrics to benchmark our model performance on the test dataset.

$Accuracy = \frac{TP}{TP+TN}$	$Precision = \frac{TP}{TP+FP}$
$Recall = \frac{TP}{TP+FN}$	$F1\text{-}score = 2 \text{ x } \frac{Precision \; x \; Recall}{Precision+Recall}$

Method	Accuracy	Precision	Recall	F1-score
DistillBERT with finetuning	48.0%	50.0%	48.0%	49.0%

Fig 5.1 Best performance achieved for 8 Classes.

Method	Accuracy	Precision	Recall	F1-score
Multinomial Naive Bayes (with BoW)	73.3%	82.0%	61.0%	54.0%

Fig 5.2 Best performance achieved for 3 Classes (Good/Average/Poor Sentiments)

Method	Accuracy	Precision	Recall	F1-score
Random Forest Classifier (with TF-IDF)	86.3%	87.0%	86.0%	86.0%
Multinomial Naive Bayes (with TF-IDF)	89.2%	89.0%	89.0%	89.0%
Logistic Regression (with TF-IDF)	90.6%	91.0%	91.0%	91.0%
DistillBERT with finetuning	93.0%	94.0%	93.0%	93.0%

Fig 5.3 Classifying into 2 Classes (Good/Bad Sentiments)

We improved on the papers by experimenting with classifying 8, 3, and 2 different classes. We found that the models generally performed best when performing binary classification (2 classes). The difference between the rating scores in this dataset is small, and the features extracted were insufficient for models to make distinctions between reviews well. The models using the TF-IDF vectorizer worked better than those using the Count vectorizer. As TF-IDF emphasises words that are important to a document but not overly common in the corpus, giving more weight to rarer, informative words and providing better insights into the document's content. Text embeddings from DistillBERT outperformed frequency-based embedding approaches (TF-IDF and Count Vectorizer) since these embeddings also consider nuances and semantic meaning of words, extracting more informative features from the text. For binary classification, we achieved performance comparable to those in related studies with our ML methods, within the range of 85-91%. Our DL approach outperformed the ML benchmarks, but slightly underperformed BERT models in related studies (>95% accuracy for most NLP benchmark tasks). This is because DistillBERT sacrifices performance for faster computation. Our proposed best model is useful for applications where computation resources for training and inference are limited since it is less computationally expensive and still provides good performance.

References

[1] Rahman A, Hossen MdS (2019) Sentiment analysis on movie review data using Machine Learning Approach. 2019 International Conference on Bangla Speech and Language Processing (ICBSLP). doi: 10.1109/icbslp47725.2019.201470

[2] Sahu, T. P., & Ahuja, S. (2016). Sentiment analysis of movie reviews: A study on feature selection & classification algorithms. 2016 International Conference on Microelectronics, Computing and Communications (MicroCom). doi:10.1109/microcom.2016.7522583

[3] Yoo, J.-Y., & Yang, D. (2015). Classification scheme of unstructured text document using TF-IDF and naive bayes classifier. Advanced Science and Technology Letters. https://doi.org/10.14257/astl.2015.111.50

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, pages 6000–6010.

[5] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/arXiv.1810.04805

[6] Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to Fine-Tune BERT for Text Classification? Computation and Language (Cs.CL). https://doi.org/10.48550/arXiv.1905.05583

[7] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1):1929–1958, 2014.

[8] Loshchilov, I., & Hutter, F. (2019). Decoupled Weight Decay Regularization. Machine Learning (Cs.LG). https://doi.org/https://doi.org/10.48550/arXiv.1711.05101

[9] Maneja, D. (2021). How and Why TF-IDF Works. Medium. https://towardsdatascience.com/how-tf-idf-works-3dbf35e568f0