A Comparative Study of Classification Models on the IMDb Dataset

Prateik Sinha
905584484
prateiksinha@g.ucla.edu

Anvesha Dutta
605499411
anveshadutta@g.ucla.edu

Christopher Fu
605480510
christopherfu01@g.ucla.edu

Osvaldo Valentino Aceves
505564001
oaceves@ucla.edu

Emily Tieu
605587766
emilytieu@g.ucla.edu

December 9, 2023

Abstract

This paper examines the IMDb Dataset. We compare and contrast several methods of supervised and unsupervised learning to classify the documents as positive or negative. Through this process we aim to evaluate these models on performing sentiment analysis. We also compare different embedding techniques such as Bag of Words, TF-IDF, Word2Vec, GloVe, and embeddings from pretrained transformers, and employ techniques such as dimension reduction to reduce computational complexity, avoid the curse of dimensionality, and increase interpretability.

1 Introduction

An IMDb review will include the following two components:

1. Review: Some sentences detailing the experience of the movie.

2. Sentiment: A binary rating of a review being either a positive or negative.

In this project, we are tasked with performing binary sentiment classification on an IMDb Dataset consisting of 50,000 movie reviews split into 25,000 highly polar movie reviews for training and testing. The challenges for this project were (1) Transforming Text into Numerical Data and (2) High-dimensional Features. Our study provides valuable insights into the strengths and weaknesses of different classification techniques and their trade-offs for sentiment analysis.

2 Previous Work

There exist several excellent review papers that aggregate the results of different architectures for sentiment analysis, ranging from SVMs to large ensemble models. Classifying this dataset with high accuracy is largely a solved problem in the eyes of the statistics and machine learning community.[6]

Rank	Model	Accuracy	Extra Training Data	Paper	Code	Result	Year
1	XLNet	96.21	✓	XLNet: Generalized Autoregressive Pretraining for Language Understanding			2019
2	Heinsen Routing + RoBERTa Large	96.2	×	An Algorithm for Routing Vectors in Sequences			2022
3	EFL	96.1	×	Entailment as Few-Shot Learner			2021
4	GraphStar	96.0	✓	Graph Star Net for Generalized Multi-Task Learning			2019
5	DV-ngrams-cosine with NB sub-sampling + RoBERTa.base	95.94	×	The Document Vectors Using Cosine Similarity Revisited			2022
6	DV-ngrams-cosine + RoBERTa.base	95.92	×	The Document Vectors Using Cosine Similarity Revisited			2022
7	BERT large finetune UDA	95.8	✓	Unsupervised Data Augmentation for Consistency Training			2019
8	BERT_large+ITPT	95.79	✓	How to Fine-Tune BERT for Text Classification?			2019
9	RoBERTa.base	95.79	×	The Document Vectors Using Cosine Similarity Revisited			2022
10	L MIXED	95.68	✓	Revisiting LSTM Networks for Semi-Supervised Text Classification via Mixed Objective Function			2020
11	BERT_base+ITPT	95.63	✓	How to Fine-Tune BERT for Text Classification?			2019

Figure 1: Performance of different neural networks on the IMDb dataset

The following works review the different techniques and methods involved in sentiment analysis:

1. *Text Classification Algorithms: A Survey*[1]

2. *A recent overview of the state-of-the-art elements of text classification*[4]

3. *Deep Learning–based Text Classification: A Comprehensive Review*[3]

Our research paper hopes to build on this body of work. We focus on exploring a wide range of approaches, including some unsupervised ones (KNN, K-means, LDA), rather than aiming to achieve perfect accuracy. Many of the 'lower accuracy' methods are far more computationally efficient and thus more feasible for real world deployment. Furthermore, our contribution includes comparing different styles of embeddings and showing the effect of dimension reduction techniques such as PCA.

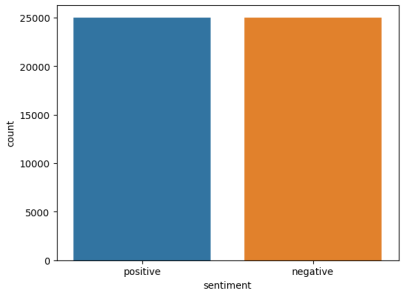


Figure 2: Balanced Dataset for both positive and negative groups

Figure 3: WordCloud of Lemmatized Words

3 Preprocessing Steps

We began with a csv file that contained 50,000 entries for the IMDb reviews described into two columns: review and sentiment. To also consider the lexicon dictionary that contained two text files of possible positive and negative sentiment words, we parsed through each review and only retained words that were also present in either the positive or the negative word list, hence creating a new column in the dataset with the much-reduced reviews. By changing the amount of words that go into preprocessing, it influences the model's ability to understand relationships and improve its performance. We tested both word banks to see how changing the complexity affected the accuracy of the model performance and chose the function class with a middle bias-variance tradeoff.

3.1 Exploratory Data Analysis

On performing some exploratory data analysis, we found out that the dataset was balanced and the positive and negative sentiment groups had 25,000 data points each. Moreover, the average length of a review within each group was also almost equal. This meant we wouldn't have to worry about balancing the dataset.

Sentiment	Mean Review Length (words)
Negative	1294.06436
Positive	1324.79768

3.2 Data Cleaning

To clean the original reviews column, we created individual functions to remove any HTML strips, URL data, square brackets, or special characters and retained only lowercase alphabets and numbers. We also expanded on some contracted words like "'ve" to 'have' and "'ll" to "will", etc. Once the text in the review column was cleaned, we proceeded to normalize the text through lemmatization. Lemmatization is a linguistic and natural language processing (NLP) technique that involves reducing words to their base or root form. The base form of a word is called the "lemma." In lemmatization, different forms of a word, such as plurals or verb conjugations, are transformed into a single common base form. The purpose of lemmatization is to normalize words so that variations of the same word are treated as a single entity. This helps in reducing the dimensionality of the feature space and improving the performance of text analysis tasks, such as sentiment analysis. First, we used the TokTok tokenizer to break down each review into words as tokens and further reduced each word to its base form using the Word Net lemmatizer. Next, we removed all stop words in the English language that occurred in the reviews. Finally, we used the Label Binarizer to convert the sentiment column to just values of 1's and 0's from positive and negative values.

The 'wordcloud' package played a pivotal role in visually representing the most prominent terms within the processed text. By assigning varying font sizes to words based on their frequency of occurrence, the resulting word cloud provides an intuitive and engaging snapshot of the sentiments expressed in the analyzed content.

3.3 Feature Engineering

Once the data was initially cleaned, we split the data into test and train sets with the test set having 20% of the data which is 10000 entries. We used two types of embeddings for vectorization to use for our models: Bag of Words and TF-IDF. For both embeddings, we considered unigrams, bigrams, and trigrams to maximize our understanding of the semantic context of the words. The data in the raw form was quite huge in terms of dimensions since the BoW embeddings had 7248842 features and the TF-IDF embedding had 6633778 features. We employed different dimension reduction techniques depending on the model like Principal Component Analysis (PCA), Non-negative Matrix Factorization (NMF), Singular Value Decomposition (SVD), and t-Distributed Stochastic Neighbor Embedding (t-SNE). These techniques helped us to trans-

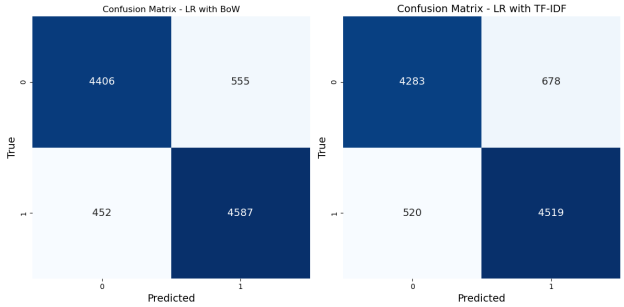


Figure 4: Confusion Matrix for Logistic Regression with both BoW and TF-IDF Embeddings

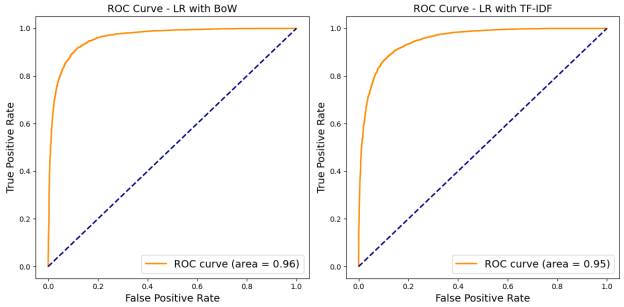


Figure 5: ROC Curves for Logistic Regression with both BoW and TF-IDF Embeddings

form high-dimensional feature representations like the ones resulting from Bag of Words and TF-IDF embeddings into lower-dimensional representations. Moreover, using the opinion lexicon also massively reduced the feature-space dimension for both types of embeddings. Bag of Words had 6760 features and TF-IDF had 926175 features.

4 Supervised Classification

4.1 Logistic Regression

Logistic Regression can be used in this context since it is a commonly used, fairly simple model used for binary classification. We fit logistic regression with both types of word embeddings, Bag of Words and TF-IDF. Oftentimes, predictive models perform too well on the training data but falter when dealing with unseen data. To prevent this phenomenon called overfitting, we applied L2 regularization to the logistic regression models since L1 regularization leads to sparsity. We also did hyperparameter tuning for the only hyperparameter that is majorly effective in Logistic Regression: C, which represents the inverse of the regularization strength. A smaller C implies a stronger application of regularization and a simpler model with more coefficients closer to 0 while a larger C indicates a weaker regularization application and a more complex model with more larger values for the coefficients. Hyperparameter tuning gave 1 as the optimized C value used in our models. We attempted to fit models on both the original reviews, the reduced reviews based on the opinion lexicon, and reviews after applying PCA, NMF, and t-SNE. We got the following results for the model considering the original review column and without any dimension analysis:

The following table summarizes all the accuracies:

Type of Model	All Words	Lexicon Dictionary	W/ Dimension Reduction
Bag of Words	0.8993	0.8476	0.8994
TF-IDF	0.8802	0.8495	0.8802

4.2 Random Forest

Random Forest consists of an ensemble of decision trees, each contributing to the final prediction. While interpreting individual decision trees might be complex, analyzing aggregated decision paths can offer insights into the model's sentiment analysis. Examining how the model traverses different words and their combinations in making predictions allows us to discern patterns and connections between specific expressions and sentiments. Understanding decision paths could reveal how the model recognizes sentiment shifts based on nuanced language.

When comparing the performance of bag of words and TF-IDF in sentiment analysis, the slightly better performance of bag of words can be understood in the context of the task's requirements. Bag of words captures the essence of sentiment through the prevalence of specific words. TF-IDF, while effective in capturing the uniqueness of words, might be less sensitive to the importance of frequently occurring sentiment-specific words. In scenarios where the task heavily relies on identifying specific sentiment-related terms, the simplicity and directness of the bag of words approach might contribute to its slightly better performance.

The unexpected lower performance when adjusting hyperparameters could be attributed to the default balance of the model. Fine-tuning hyperparameters might lead to a suboptimal configuration, causing the model to become overly specialized to the training data. It shows the importance of striking a balance that promotes model adaptability while avoiding overfitting.

4.3 K-Fold Cross Validation on Naive Bayes Classifier

The application of k-fold cross-validation serves as a robust technique to assess and enhance the performance of our Naive Bayes sentiment analysis model. With the choice of k = 5 folds, the training dataset on the Lexicon Dictionary Bag of Words was systematically partitioned into five subsets, with each subset

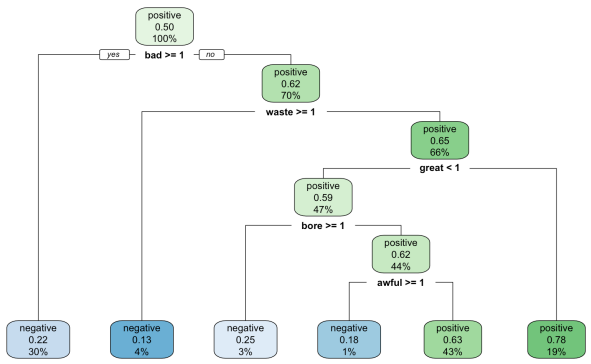


Figure 6: Best Performing Tree within the Random Forest for Bag of Words

serving as both a testing and validation set across distinct iterations. This iterative process allowed for a comprehensive evaluation of the model's generalization capabilities, mitigating the risk of overfitting or underfitting to a specific set of data.

Through k-fold cross-validation, we observed a slight enhancement in the model's performance metrics compared to a traditional single split of the data. The approach not only provided a more stable and reliable estimation of the model's accuracy but also yielded a reduction in variance by averaging performance metrics across multiple folds.

We will showcase the model train on the Bag of Words from the Lexicon Dictionary. Surprisingly, this dataset performed better than the Lemmatized words. One possible reasoning for this is that by choosing a smaller vocabulary, you can mitigate data sparsity, as the model is less likely to encounter rare words with little impact. This can result in a more stable and generalizable model.

Control	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Test
0.7925	0.7918	0.7982	0.7995	0.7933	0.7969	0.7992

A final test on the testing dataset gave us an accuracy of 0.7992. Compare this to the control test of only a single partition which gave us an accuracy of 0.7925. The validation process not only bolsters our confidence in the model's generalizability but also contributes to the overall reliability and credibility of our sentiment analysis framework.

4.4 Deep Learning Methods

Deep learning methods provide far and away the best performance for binary classification problems such as this one, albeit at the cost of significantly longer training times and higher computational cost. There are several models that have already achieved accuracies in the high 90s range for this particular dataset as we saw in the related works section.

While the previous work on this topic essentially makes this a solved problem, there is still lots of diversity and variability between neural network architectures to explore. A study of binary classification techniques could not be considered comprehensive without comparing and contrasting different deep learning models.

Our research paper focuses on building and training all the neural network architectures for sentiment analysis from scratch and running them on the same hardware (A single Nvidia Laptop RTX 3060 @ 130W). This ensures that all models are trained on an even playing field, and thus provides a fair relative comparison of their performance.

Note that all the below methods use GloVe vector embeddings and word level tokenization. Code can be found at https://github.com/Prateik-11/STATS101C_project

4.4.1 Transformer

We use a transformer-encoder model to capture the latent features of the documents and use them to classify the sentiment. All the top-performing models on public benchmarks of the IMDb dataset are based on transformers, and they have been the state of the art for various NLP and vision tasks in recent years. Our implementation has relatively few parameters due to hardware constraints.

Tranformers process the entire sequence simultaneously, allowing them to utilize the parallelizing capabilities of modern GPU hardware to speed up computation. Their self attention layer allows tokens in the sequence to attend to each other, making them expressive in the forward pass. The are also easy to optimize using first degree optimizers such as gradient descent. This is a massive consideration, considering that modern neural networks are unable to utilize 2nd or higher degrees of optimization due to their computational complexity. [7]

Architecture of transformer model:

```
TransformerEncoder(
  (embedding): Embedding(400002, 50)
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0.02)
  )
  (transformer_encoder): TransformerEncoder(
    (layers): ModuleList(
      TransformerEncoderLayer x3 (
        (self_attn): MultiheadAttention(
          (out_proj):
              NonDynamicallyQuantizableLinear(
            in_features=50, out_features=50
          )
        )
        (linear1): Linear(in_features=50,
            out_features=768)
```

```
      (dropout): Dropout(p=0.02)
      (linear2): Linear(in_features=768,
          out_features=50)
      (norm1): LayerNorm((50,), eps=1e-05)
      (norm2): LayerNorm((50,), eps=1e-05)
      (dropout1): Dropout(p=0.02)
      (dropout2): Dropout(p=0.02)
    ) ) )
  (fc1): Linear(in_features=50, out_features=16)
  (relu): ReLU()
  (fc2): Linear(in_features=16, out_features=1)
  (sigmoid): Sigmoid()
)
```

Hyperparameters for transformer model:

```
'batch_size':16,
'lr':0.0001,
'n_heads': 5,
'dropout_prob': 1/50,
'hidden_size':768,
'encoder_layer_cnt':3,
'hidden_size_2': 16,
```

4.4.2 RNN (*LSTM*)

Recurrent Neural Networks, or RNNs, had been the state of the art for NLP before transformers were introduced by Vaswani et al in 2017. They take each token as an input one at a time, feeding their output back into themselves after each iteration/cell. This property is what makes them recurrent. While processing sequences one token at a time seems like a natural fit for text classification, this architecture has some drawbacks.

1. They are very slow to train since their recurrent nature means they have to process tokens one at a time and they cannot be parallelized.
2. They have trouble 'recalling' early tokens in very long sequences, i.e., the longer the sequence the higher the likelihood that they will only attend to the final few words in it
3. They can be very hard to optimize due to the exploding/vanishing gradient problem and often require fixes such as gradient clipping

The problem with recalling older tokens is somewhat alleviated in improved versions of RNNs such as the LSTM (Long Short Term Memory) architecture or the GRU (Gated Recurrent Unit) which we will be using. One advantage of the RNN is that it takes significantly less data to train than a transformer.

Architecture of LSTM model:

```
LSTMEncoder(
  (embedding): Embedding(400002, 50)
  (lstm): LSTM(50, 256, num_layers=2)
  (fc1): Linear(in_features=256,
      out_features=128)
  (fc2): Linear(in_features=128, out_features=1)
  (sigmoid): Sigmoid()
)
```

Hyperparameters for LSTM model:

```
'batch_size':32,
'lr':1e-4,
'dropout_prob':2e-1,
'hidden_size':256,
'lstm_unit_cnt':2,
)
```

4.4.3 CNN

CNNs are typically used for image processing and not for NLP, but they can be used for either. Their ability to find patterns by passing convolutional filters over data has been shown to be useful for extracting information from text embeddings.

Architecture of CNN:

```
CNNEncoder(
  (embedding): Embedding(400002, 50)
  (cnn): Conv1d(50, 32, kernel_size=(93,),
      stride=(1,))
  (fc1): Linear(in_features=256,
      out_features=128)
  (fc2): Linear(in_features=128, out_features=1)
  (sigmoid): Sigmoid()
)
```

Hyperparameters for CNN model:

```
'batch_size':32,
'lr':1e-4,
'dropout_prob':2e-1,
'hidden_size':256
```

4.4.4 Results

We train these models not only for the IMDb dataset, but also for two additional datasets: the Yelp and Quora datasets.

1. Yelp: The Yelp-2 dataset comprises of business reviews from Yelp, where positive and negative instances are two distinct classes. It encompasses 5,261,668 reviews, exhibiting a roughly even distribution of positive and negative feedback.
2. Quora: The Quora dataset is from a Kaggle competition to classify Quora questions as either sincere or insincere. The dataset, consisting of 1,306,122 samples, poses a significant challenge

due to a substantial class imbalance, with a notably low number of insincere examples.

We want to benchmark these models on their ability to perform sentiment analysis. To do so properly we need to get an idea of how they perform in various scenarios, hence our decision to use multiple datasets for this task. A future extension of our work would be to test whether performing transfer learning or jointly learning multiple datasets would lead to better performance in our models.

Dataset	Accuracy	F1	AUC ROC
IMDB	0.8309	0.8352	0.9139
Yelp	0.9093	0.9138	0.9299
Quora	0.9414	0.2845	0.8634

Table 1: Performance Metrics for Transformer (Encoder)

Dataset	Accuracy	F1	AUC ROC
IMDB	0.7818	0.7590	0.8842
Yelp	0.8365	0.8332	0.9159
Quora	0.9500	0.5604	0.9339

Table 2: Performance Metrics for RNN (LSTM)

Dataset	Accuracy	F1	AUC ROC
IMDB	0.7823	0.7750	0.9231
Yelp	0.7908	0.8455	0.8559
Quora	0.9319	0.2531	0.8414

Table 3: Performance Metrics for CNN

The Transformer model displays higher accuracy on the IMDb dataset as well as the Yelp dataset. However, for the Quora dataset, the RNN model performs slightly better. For IMDb and Yelp, the Transformer also has the highest F1 score. For the Quora dataset, RNN has the highest F1 score. All 9 AUC ROC scores are relatively similar, with no clear patterns across datasets or across models.

The IMDB and Quora datasets achieve accuracies of greater than 90% on state of the art models, as seen in the paper 'Deep Learning–based Text Classification: A Comprehensive Review'. Given the difference in the size of our models and the capabilities of our hardware, we believe our results were quite good, achieving high accuracy on these tasks with fewer parameters and simpler architectures.

Among the datasets we utilized, the Quora dataset is the most imbalanced, where only around 8% of examples belong to one of the two classes. In contrast, both the IMDB and Yelp datasets exhibit a more equitable distribution, with the Yelp dataset boasting a significantly larger sample size compared to the IMDB dataset. This distributional variation is reflected in the performance metrics, with models achieving notably high accuracy on Quora but registering a low F1 score due to diminished precision and recall.

The Quora dataset, originally sourced from a Kaggle competition that evaluated models based on their F1 score, witnessed top-performing models reaching approximately 0.7 F1 score. The winning entry involved utilizing stacks of multiple embeddings and multiple RNN layers. The efficacy of RNNs in this context is also mirrored in our models, with the LSTM model attaining the highest F1 score among the models tested on the same dataset.

It's worth noting that the challenges associated with training transformers from scratch may contribute to our models' performance, as transformers typically demand substantial amounts of data. Unfortunately, due to hardware constraints, we were unable to explore transfer learning using BERT or a language model (such as Mistral, LLaMA, GPT) to validate this hypothesis.

5 Unsupervised Classification

5.1 KNN

Before using KNN, we need to generate vector embeddings for the documents in our data set. We try a number of different strategies for generating embeddings in order to evaluate which one is capable of capturing the most relevant semantic information about our documents in their respective latent spaces.

Using PCA we reduce the dimensions of the document embeddings, otherwise methods such as KNN would be unfeasible due to the sparse feature space of high-dimensional data and the curse of dimensionality.

5.1.1 paraphrase-MiniLM-L6-v2

These embeddings are learnt by the paraphrase-MiniLM-L6-v2 model in the sentence transformers library on HuggingFace. This model is based on the BERT architecture and learns semantic meaning by mapping entire sentences to a vector space and back-propagating through a transformer-based architecture training on NLP tasks such as next word prediction and filling in missing words.

The embeddings learnt by this method have 384 dimensions. Reducing them with PCA gives us the following plot of the number of variance explained vs the number of components:

Based on the plot, we can see that the amount of variance explained starts to diminish strongly after 10 components. Since having fewer dimensions could increase the performance of the KNN algorithm, we perform a grid search for n = 1 to 10 in order to find the optimal trade-off

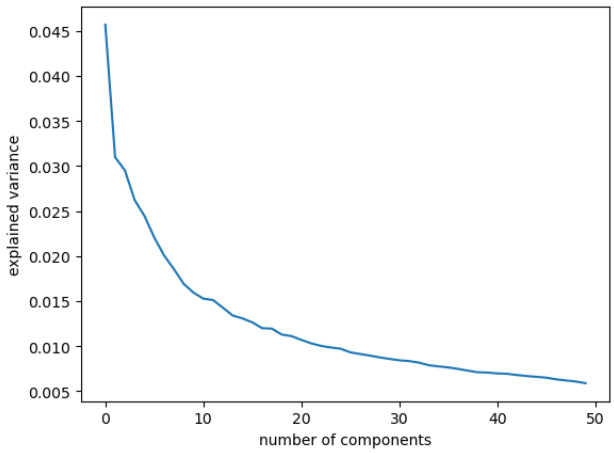


Figure 7: Variance Explained vs. No. Components for paraphrase-MiniLM-L6-v2

	1	3	5	7	9	11	13	15	17	19	21
1	0.5019	0.508898	0.514097	0.519696	0.514897	0.525095	0.527894	0.531094	0.525095	0.531094	0.534493
2	0.531494	0.54929	0.54809	0.554889	0.558288	0.560688	0.559288	0.559888	0.560488	0.562088	0.564887
3	0.467506	0.457708	0.45111	0.454589	0.44951	0.44771	0.445111	0.446111	0.446911	0.447111	0.44971
4	0.446511	0.421316	0.413917	0.407918	0.403519	0.407119	0.404119	0.402519	0.39972	0.403319	0.404519
5	0.446711	0.422116	0.414517	0.411918	0.408518	0.406519	0.409518	0.411518	0.408318	0.409318	0.410118
6	0.593081	0.60248	0.614477	0.616677	0.623475	0.624475	0.621876	0.621076	0.617477	0.621476	0.622276
7	0.455709	0.443511	0.440512	0.436913	0.434513	0.431314	0.424715	0.430314	0.428514	0.429314	0.433313
8	0.45151	0.439512	0.435913	0.430314	0.424915	0.427315	0.423515	0.421516	0.423515	0.423515	0.423115
9	0.456509	0.442312	0.44751	0.443511	0.443311	0.439912	0.438512	0.439312	0.436513	0.436713	0.435513
10	0.470706	0.454509	0.456109	0.45171	0.454309	0.45051	0.44931	0.44811	0.446511	0.445111	0.443111

Figure 8: No. components in PCA on y-axis, K for KNN on x-axis [paraphrase-MiniLM-L6-v2 KNN Accuracy]

The best accuracy found using this method/embeddings is **0.623475**, with 6 components from the PCA selected and the 9 nearest neighbors being used to vote.

5.2 K-Means Clustering

To generalize the approach to datasets without labels, we can use the same preprocessing used for KNNs and apply clustering algorithms to discover latent clusters within our data. By splitting into two clusters and measuring the rate of observing positive/negative labels in each we can see if the model will naturally learn to distinguish between positive and negative examples

5.2.1 paraphrase-MiniLM-L6-v2

Since these embeddings were not trained for sentiment analysis but rather for capturing the semantic meaning behind entire sentences, it is unlikely that this approach would naturally partition the data into groups that align with our labels. This is exactly what we see:

5.3 Latent Dirichlet Allocation

The Latent Dirichlet Allocation (LDA) generative probabilistic model is specialized in topic modeling for discovering abstract topics. In other words, the objective of LDA is to find topics for which a document belongs to based on the words within the document. Two components of LDA are (1) the words belonging to a document, and (2) the words belonging to a topic (i.e. the probability of words belonging into a topic). The latter requires computation which is accomplished by an algorithm: (1) For each document, randomly assign each word in the document to one of k chosen topics (2) For each document d and word w in the document, compute (a) the proportion of words in document d that are assigned to topic t (p(topic t — document d), and (b) the proportion of assignments to topic t over all documents originating from the word w (p(word w — topic t). We can then calculate the probability for the word w belonging to topic t as the product of these conditional probabilities. [2]

Pertaining to this project, we pre-trained the LDA model on both Bag-of-Words and TF-IDF, which resulted in the model producing topics for each word within each document. Shown in Figures 8-9 is a preview of the extracted topics and terms after applying the model.

After training our LDA model on our dataset, we used a classifier known as AdaBoost to evaluate the performance of our model. AdaBoost, short for Adaptive Boosting, is a boosting technique used as an ensemble method in machine learning in which the weights are re-assigned to each instance, with higher weights assigned to incorrectly classified instances. [5] The following results are shown in Figures 10-11.

This may be the result of the LDA model making incoherent topics which could either be due to an inherent flaw in the model itself or some noise in the data. AdaBoost is comparably a better classifier than

No. PCA Components	1	2	3	4	5
Cluster 1	0.574053	0.575665	0.435197	0.436197	0.435274
Cluster 2	0.431961	0.430844	0.571814	0.571016	0.571011

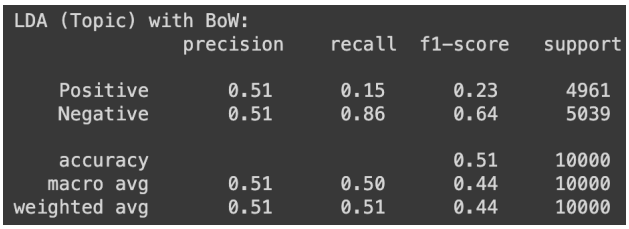
Figure 9: Percentage of positive sentiment documents in each cluster after K-mean clustering with different numbers of PCA components

```
Topic 0:  ['much better', 'ever seen', 'special effects', 'looks like', 'see movie', 'one best', 'new york']
Topic 1:  ['ever seen', 'one best', 'good movie', 'first time', 'waste time', 'special effects', 'even though']
Topic 2:  ['ever seen', 'one best', 'low budget', 'see movie', 'even though', 'special effects', 'look like']
Topic 3:  ['special effects', 'looks like', 'ever seen', 'even though', 'see movie', 'watch movie', 'one best']
Topic 4:  ['even though', 'ever seen', 'special effects', 'one best', 'movie would', 'new york', 'waste time']
```

Figure 10: Extracted Topics and Terms from the LDA model for Bag of Words

```
Topic 0:  ['good movie', 'waste time', 'looks like', 'love story', 'special effects', 'low budget', 'look like']
Topic 1:  ['special effects', 'watch movie', 'new york', 'saw movie', 'watching movie', 'good movie', 'waste time']
Topic 2:  ['waste time', 'saw movie', 'special effects', 'good movie', 'watch movie', 'im sure', 'looks like']
Topic 3:  ['waste time', 'special effects', 'looks like', 'film really', 'pretty good', 'high school', 'look like']
Topic 4:  ['watch movie', 'low budget', 'looks like', 'special effects', 'good movie', 'great movie', 'bad movie']
```

Figure 11: Extracted Topics and Terms from the LDA model for TF-IDF



LDA (Topic) with BoW:

	precision	recall	f1-score	support
Positive	0.51	0.15	0.23	4961
Negative	0.51	0.86	0.64	5039
accuracy			0.51	10000
macro avg	0.51	0.50	0.44	10000
weighted avg	0.51	0.51	0.44	10000

Figure 12: Table of Model Performance for Bag of Words

LDA (Topic) with TF-IDF:

	precision	recall	f1-score	support
Positive	0.51	0.13	0.20	4961
Negative	0.51	0.88	0.64	5039
accuracy			0.51	10000
macro avg	0.51	0.50	0.42	10000
weighted avg	0.51	0.51	0.42	10000

Figure 13: Table of Model Performance for TF-IDF

most weak classifiers but it can also be resource intensive.

6 Results

Table 4: Supervised Classification Results

Method	**Accuracy (BoW)**
Logistic Regression	89.93% (BoW) 88.02% (TF-IDF)
Random Forest	86.34% (BoW) 85.65 % (TF-IDF)
K-Fold CV on Naive Bayes	79.92% (BoW) 78.04% (BoW w/o K-Fold CV)
Deep Learning Methods	**Accuracy (GloVe)**
Transformer	83.09%
RNN (LSTM)	78.18%
CNN	78.23%

Table 5: Unsupervised Classification Results

Method	**Accuracy**
KNN	62.35%
K-Means Clustering	57.57%
Latent Dirichlet Allocation (LDA)	51%

Unsupervised seems to perform worse than supervised learning which makes sense since the models had less data to operate on and were not explicitly taught to distinguish based on review sentiment. Out of the supervised models, logistic regression had the highest accuracy of 89.93%, closely followed by random forest with an accuracy of 86.34%, both having used the Bag of Words embedding. Their performance being the best out of all models could be due to several reasons. The data could be successfully separated by a linear decision boundary into positive and negative sentiments which could have boosted logistic regression's performance. Random forest is an ensemble method that aggregates results from multiple decision trees leading to improved accuracies and better generalization of the model. The model has reduced overfitting too due to its bagging algorithm and hence had high results for our dataset. Moreover, some other models that we tried out like KNN and SVM require intensive hyperparameter tuning without which model performance may be suboptimal.

The Naive Bayes model showed the lowest accuracy when using BoW. Since the Naive Bayes model simplifies assumptions, it may not have accurately captured complex relationships in the dataset compared to the more sophisticated models. Applying K-fold cross-validation slightly increased the accuracy score. This is possibly due to the model not being suitable for the data, hence K-fold cross-validation provides better generalization by evaluating the performance across different subsets of the data. Since the model is also generally sensitive to data quality, using the lexicon dictionary may reduce the noise in the data. Hence the simplicity and efficiency of the lexicon approach can alleviate any misleading or noisy data and reduce any complex relationships that could not be picked up when using the full data since Naive Bayes assumes that the attributes are conditionally independent.

The deep learning-based methods using general-purpose semantic embeddings were outperformed by simpler methods such as logistic regression using Bag of Words. Considering that our task of classifying documents as positive or negative is largely dependent on whether certain words are present in the review, the information provided by BoW was more relevant than GloVe or Word2Vec. Thus we conclude that extracting relevant information from the document leads to higher gains in performance than simply increasing the size and scale of the model. While it is possible to achieve highly competitive results using large transformer-based approaches as shown in related works [6], we can do so with far fewer parameters by carefully and deliberately choosing our embeddings.

LDA for both Bag-of-Words and TF-IDF had an underwhelming precision of 51%, as well as a recall of around 15% for classifying positive labels and 88% for classifying negative labels, suggesting that LDA is unable to properly generalize information gained from the training data to classify as positive words.

Based on our holistic observations of our findings, we see that Logistic Regression is the best model for accurate sentiment classification for this dataset given the compute we have access to. To further optimize the results, one could experiment with different methods of cross-validation like leave-one-out and stratified, as well as search methods like randomized search and Bayes optimization for hyperparameter tuning when it comes to the parametric models like decision trees,

tree-based ensemble methods, KNN and SVM. Additionally, the algorithms of bagging and boosting could be employed to optimize the bias-variance trade-off, reduce overfitting, and better generalize the model for unseen data. Lastly, since our current dataset was balanced, it may be unrepresentative of real-world data since datasets are naturally imbalanced and training and fitting models using an artificially-balanced dataset may lead to poor performance in the real world.

7 Future Work

1. Transfer Learning: The neural networks were all trained on one dataset at a time. We may be able to leverage gains in performance from jointly learning multiple tasks however we were not able to explore this because of time constraints.

2. Contrastive Learning: This deep learning technique focuses on learning an embedding function for your data as well as for each of your classes such that the cosine similarity is minimal between the embeddings of a data point and its corresponding class, and maximal for a data point and any other class. This is commonly used in siamese networks, for tasks like facial recognition where the model must generalize to faces outside of its training distribution. We could employ this technique to make a general sentiment analysis classifier that works for any dataset. Models such as OpenAI's CLIP are able to extend this technique to multi-modal space by mapping images to the same latent space as text. Thus, we could theoretically build a sentiment analysis classifier for not only text but also pictures. While we got started working on this, unfortunately we did not have enough time to develop this approach.

We would also like to do more rigorous hyperparameter tuning for some of the methods we described.

References

[1] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4), 2019. 1

[2] Ria Kulshrestha. A beginner's guide to latent dirichlet allocation(lda). 7

[3] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning–based text classification: A comprehensive review. *ACM Comput. Surv.*, 54(3), apr 2021. 1

[4] Marcin Michał Mirończuk and Jarosław Protasiewicz. A recent overview of the state-of-the-art elements of text classification. *Expert Systems with Applications*, 106:36–54, 2018. 1

[5] Anshul Saini. Adaboost algorithm: Understand, implement and master adaboost. 7

[6] Various Authors. Sentiment analysis on imdb: Leaderboard, 2023. [Online; accessed 8-December-2023]. 1, 8

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 4