



CSE674 - TEXT ANALYTICS

ASSIGNMENT I

INSTITUTE OF BUSINESS ADMINISTRATION - IBA

Sentiment Analysis with Various Approaches

Submitted By:

Bilal Naseem - ERP ID: 13216

Hammad Hadi Khan - ERP ID: 14278

Nehal Ahmed - ERP ID: 25751

Date: February 25, 2024

Abstract

In our project, we conducted comprehensive sentiment analysis using various approaches and benchmarked our results against the distilbert-base-uncased-finetuned-sst-2-English model. Firstly, we explored lexicon-based approaches by experimenting with different lexicon-based libraries to identify the most accurate model without requiring any training. Subsequently, we delved into classical machine learning approaches, where we preprocessed the text data by tokenization, stemming/lemmatization, POS tagging, and stopword removal.

We then vectorized the text data using TF-IDF or CountVectorizer and trained different machine learning models such as Naive Bayes, Random Forest, and k-NN on the processed training dataset. The accuracy of these models was evaluated on the test data.

Next, we ventured into customized word embeddings, training different machine learning models like Naive Bayes, Random Forest, and k-NN on these embeddings. Additionally, we fine-tuned three different pre-trained language models (PLMs), namely DistilBERT, RoBERTa, and GPT-2, for sentiment analysis on our dataset. Finally, we explored hybrid approaches, combining lexicon-based, machine learning, embedding, and PLM-based methods to achieve improved performance on the test data. Our project provides valuable insights into the effectiveness of various sentiment analysis techniques and offers a comprehensive comparison against the pre-trained language model.

Contribution of each member

1. **Hammad Hadi Khan:** Task 1, 2, 3
2. **Nehal Ahmed:** Task 3
3. **Bilal Naseem:** Task 4, 5

Code

- All Notebooks: ([Github Repo](#))

Contents

1	Introduction	3
1.1	About the Dataset	3
1.1.1	Histogram of Review Lengths	4
1.1.2	Top N Most Frequent Words	5
1.2	Feature Engineering	5
2	Lexicon-based Approach	6
3	Classical Machine Learning Approaches	7
4	Customized Word Embeddings	9
4.1	Word2Vec: Skip-gram and CBOW Models	9
4.2	Skip-gram Model	10
4.3	Continuous Bag of Words (CBOW) Model	11
4.4	Gensim Implementation	11
4.5	Evaluation of Machine Learning Models	12
5	Fine-tuning Pre-trained Language Models (PLMs)	14
5.1	DistilBERT	14
5.2	RoBERTa	15
5.3	GPT-2	15
6	Hybrid Approach	16
7	Conclusion & Future Work	17
7.1	Future Work	17

1 Introduction

1.1 About the Dataset

Data Columns

The imdb dataset contains two columns; “review” and “sentiment”. The “review” column contains the textual information (input features) and the “sentiment” column contains the output labels. The task of any classifier and method implemented in this project is to correctly predict the “sentiment” given any “review” in the textual column. Hence we have to apply our data cleaning, transformation steps to the “review” column.

Bar Plot of Sentiment Distribution:

We utilize this bar plot to illustrate the distribution of sentiments (positive and negative) in our dataset, providing an overview of the balance between positive and negative reviews. It allows us to quickly grasp the proportion of positive and negative sentiments present in our dataset.

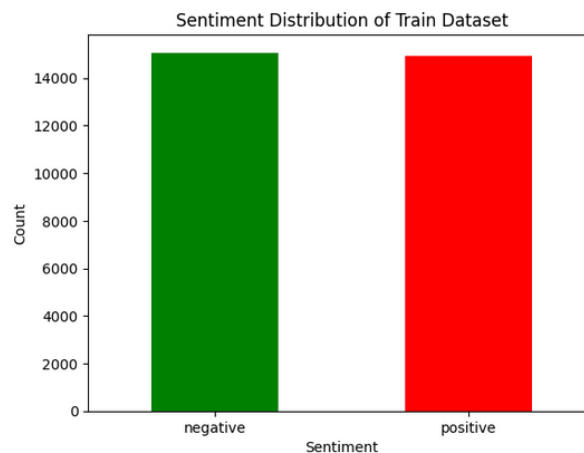


Figure 1: Bar Plot of Sentiment Distribution

Word Clouds for Each Sentiment

These word clouds visually represent the most frequent words in both positive and negative reviews, offering insights into the dominant sentiments expressed in our dataset. By showcasing the words with varying font sizes based on their frequency, the word clouds highlight the key themes and sentiments prevalent in our reviews.

1.1.1 Histogram of Review Lengths

The histogram displays the distribution of review lengths (number of words), enabling us to understand the range and frequency of review lengths in our dataset. It provides insights into the distribution of review lengths, identifying any outliers or patterns in the length of reviews.

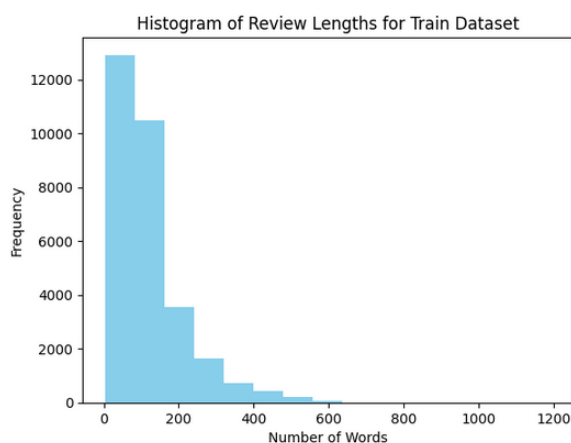


Figure 2: Histogram of Review Lengths

Box Plot of Review Lengths by Sentiment

This box plot compares the distribution of review lengths between positive and negative sentiments, revealing any differences in the length of reviews based on sentiment. It visually depicts the central tendency, spread, and potential outliers of review lengths for each sentiment category.

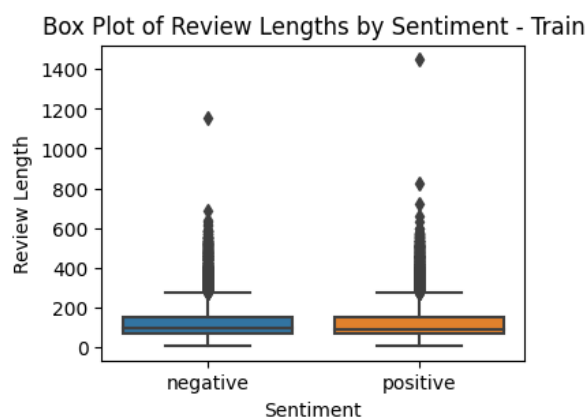


Figure 3: Box Plot of Review Lengths by Sentiment

1.1.2 Top N Most Frequent Words

We utilize these bar charts to display the top N most frequent words in our dataset, separately for positive and negative reviews, offering insights into the most commonly used words in each sentiment category. By identifying and visualizing the most frequent words, these plots help us understand the language patterns and sentiments expressed in our reviews.

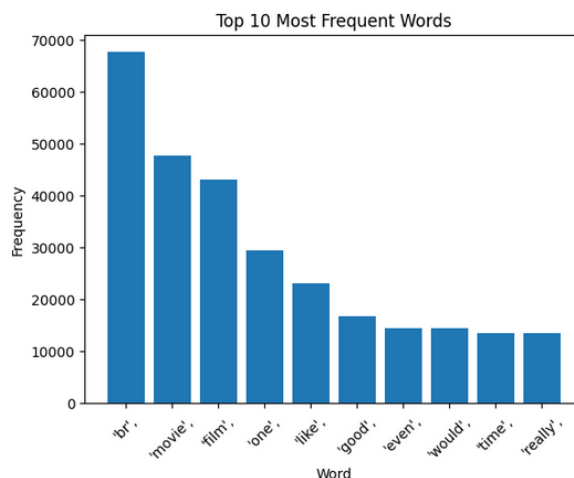


Figure 4: Top N Most Frequent Words

1.2 Feature Engineering

The text column of the dataset was cleaned using the below techniques:

1. Parsing HTML tags such as `
` etc.
2. Removing accented characters
3. Expanding contractions
4. Removing special characters
5. Removing stop words
6. Stemming and Lemmatization

The code for preprocessing can be found in the below notebook:

- Pre-processing notebook

The below scripts were also utilized in this stage:

```
1 !wget https://raw.githubusercontent.com/Puspita-111/Suven-consultants-and-technology/main/
   !contractions.py
2 !wget https://raw.githubusercontent.com/Puspita-111/Suven-consultants-and-technology/main/
   !text_normalizer.py
3
```

The dataset was cleaned using various approaches. The cleaning process aims to remove redundancies such as HTML codes, URLs, emojis, stopwords, punctuations, and expand abbreviations. These steps are essential to ensure the quality and reliability of the dataset for subsequent analysis. The dataset cleaning process involved the following steps, implemented using regular expressions (regex). In order to evaluate the pre-processing of the dataset, it was passed through **Naive Bayes** (after training on the **tfidf_vectorizer**, and on the raw **DistilBERT** model. The results are below:

Test Dataset Performance on Naive Bayes					
Dataset no.	Description	Accuracy	Precision	Recall	F1
1	Without any pre-processing	86.12%	86.74%	85.49%	86.11%
2	Stripping HTML tags & Sp. Charact	85.99%	86.22%	85.90%	86.06%
3	Removing accented characters	86.03%	86.11%	86.13%	86.12%
4	All of the above + Lemmization	86.67%	86.33%	85.15%	85.73%

Figure 5: Test Dataset Performance on Naive Bayes

Test Dataset Performance on Untrained Distilbert-uncased					
Dataset no.	Description	Accuracy	Precision	Recall	F1
1	Without any pre-processing	53.46%	57%	31%	40%
2	Stripping HTML tags & Sp. Charact	50.01%	60%	20%	40%
3	Removing accented characters	49.84%	50%	98%	66%
4	All of the above + Lemmization	53.12%	56%	31%	41%

Figure 6: Test Dataset Performance on Untrained Distilbert-uncased

The results show accuracy hasn't been affected much through pre-processing.

2 Lexicon-based Approach

We utilized **TextBlob**, a Python library offering a simple API for various text processing tasks, including sentiment analysis. It employed a combination of lexicon-based approach and machine learning techniques to assign polarity scores to text, facilitating quick and straightforward sentiment analysis.

VADER (Valence Aware Dictionary and Sentiment Reasoner) serves as a powerful tool for sentiment analysis, specifically tailored for social media text. We relied on VADER's pre-built lexicon of words and their sentiment scores, along with its rule-based system designed to handle nuanced language patterns, enabling accurate sentiment analysis of text containing emoticons, slang, and punctuation.

We employed NLTK's Sentiment Intensity Analyzer, a lexicon-based approach provided by the NLTK library for sentiment analysis. This tool enabled us to determine the sentiment intensity of text by assigning scores based on the strength of positive and negative words, allowing for nuanced analysis of sentiment.

Lexicon Based Intensity analysis	Train Data Accuracy	Test Data Accuracy
TextBlob	46.14%	69.22%
VADER	46.84%	70.23%
NLTK Sentiment Intensity Analyzer	46.63%	69.95%

Figure 7: Lexicon Based Intensity analysis

3 Classical Machine Learning Approaches

TF-IDF (Term Frequency-Inverse Document Frequency) and Count Vectorization are two commonly used techniques for converting text data into numerical vectors, which can be used as input for machine learning algorithms. In this section, we will explore and implement both vectorization strategies on our corpus. **TF-IDF Vectorization:** TF-IDF works by calculating a weight for each term in a document based on its frequency (TF) and inverse document frequency (IDF). The TF-IDF weight increases with the frequency of the term in the document, but is offset by the frequency of the term in the entire corpus. The formulation is as follows:

$$TF - IDF = TF(t, d) * IDF(t, D) \quad (1)$$

where: $TF(t, d)$ represents the frequency of the term t in document d . $IDF(t, D)$ represents the inverse document frequency of term t in the corpus D .

Count Vectorization:

Count Vectorization is a simpler technique that represents each document as a vector of term frequencies. Each element of the vector corresponds to the frequency of a particular term in the document. The dimensionality of the vector is equal to the vocabulary size of the corpus.

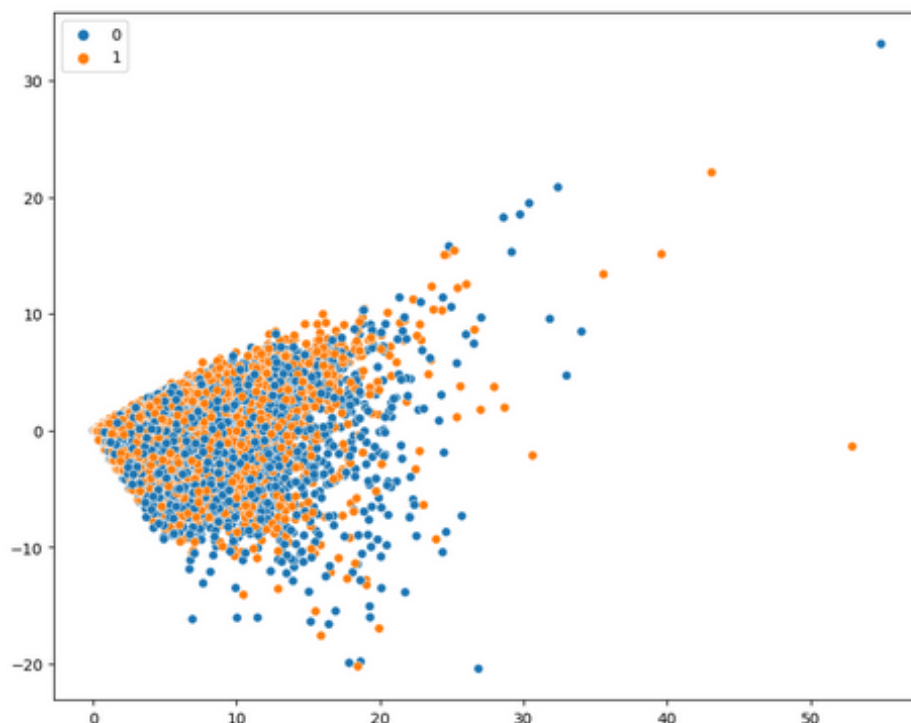


Figure 8: Count Vectorization

Implementation:

We implemented **TF-IDF** and **Count Vectorization** on our corpus and examined the dimensionality of the resulting vectors.

Processing

We converted the dataset into **TF-IDF** to analyze sentiment in IMDb movie reviews. The TF-IDF Vectorizer converts text into a matrix, representing each document and unique word in the corpus. The training and testing data is transformed into **TF-IDF** vectors using the `fit_transform()` method. Target labels are extracted from the respective DataFrames.

We used three classifiers to initialize and train using the **TF-IDF** transformed training data:

- Multinomial Naive Bayes (clf1)
- k-Nearest Neighbors (clf2)

- Random Forest (clf3)

Each classifier is trained with the **TF-IDF** vectors and corresponding target labels. The trained classifiers predict sentiment labels for the test data using the `predict()` method. The results were as follows:

- Multinomial Naive Bayes classifier achieved the highest accuracy of **85.87%**
- Random Forest classifier performed well with **84.60%** accuracy.
- k-Nearest Neighbors classifier had the lowest accuracy, around **73.72%**.

4 Customized Word Embeddings

Word embeddings are numerical representations of words in vector space. They capture the semantic meaning of words by representing them as dense vectors, where similar words are located closer to each other in the vector space. Word embeddings are widely used in natural language processing tasks, including sentiment analysis, text classification, and machine translation.

1. **Similarity Measurement:** Word embeddings enable us to measure the similarity between words based on their vector representations. This can be done using metrics such as cosine similarity or Euclidean distance. Similar words have vectors that are close to each other in the vector space.
2. **Distance Measurement:** Similar to similarity measurement, distance metrics such as Euclidean distance or cosine distance can be applied to word vectors to measure the distance between words. Words that are semantically similar tend to have smaller distances between their vectors.
3. **Numerical Transformations:** Word embeddings allow for numerical transformations of word vectors, such as addition, subtraction, or multiplication. These operations can yield interesting insights, such as finding analogies or completing analogical reasoning tasks.

4.1 Word2Vec: Skip-gram and CBOW Models

Our group explored **Word2Vec**, a popular word embedding technique, which learns distributed representations of words in a continuous vector space. By representing each word as a dense vector, Word2Vec captures semantic relationships between words, enabling various natural

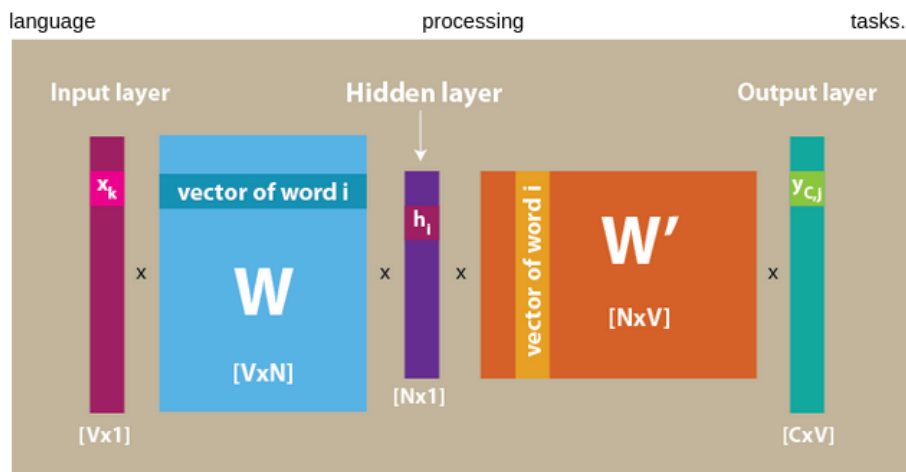


Figure 9: Word2Vec

4.2 Skip-gram Model

Our group delved into the **Skip-gram** model, one of the architectures used by Word2Vec. In this model, we aimed to predict context words given a target word. By maximizing the probability of context words given the target word, the Skip-gram model effectively captured syntactic relationships between words, particularly in scenarios with large vocabularies.

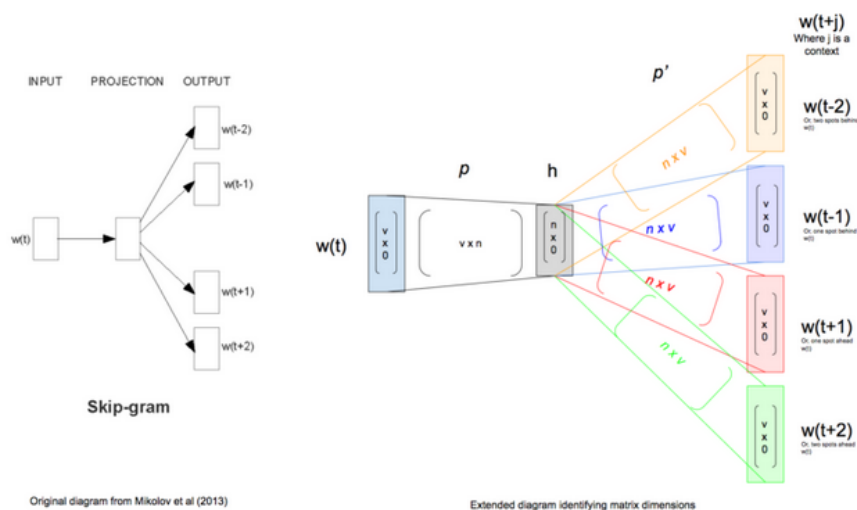


Figure 10: Skip-gram Model

4.3 Continuous Bag of Words (CBOW) Model

We also explored the **Continuous Bag of Words (CBOW)** model, another architecture used by Word2Vec. In this model, our objective was to predict the target word given a context of surrounding words. By maximizing the probability of the target word given the context words, the **CBOW model** effectively captured semantic relationships between words, especially when the context provided rich information about the target word.

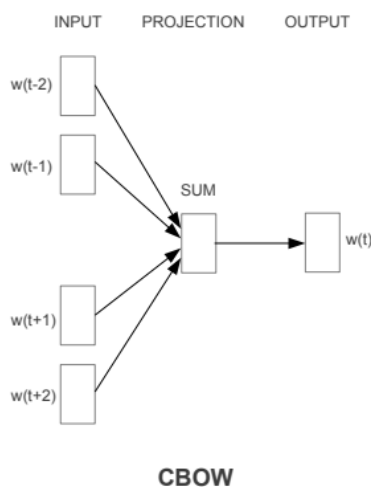


Figure 11: (CBOW) model

4.4 Gensim Implementation

Our group utilized **Gensim**, a Python library, to train and utilize **Word2Vec** models seamlessly. Gensim provided convenient classes and functions to build, train, and utilize **Word2Vec** models, abstracting away the implementation details and enabling us to focus on our specific tasks without the need to worry about the underlying algorithms.

We used cosine distance measurement to determine the closeness of two word vectors in a vector space. It measures the cosine of the angle between the vectors, indicating the similarity between the words they represent. When plotting word vectors using matplotlib, the resulting graphs often resemble **electrocardiogram (ECG)** diagrams. This is due to the analysis of each word based on the number of words in its surrounding context.

We then used **PCA** transformation to reduce the dimensionality of embedded words from a Word2Vec model. It initialized a Word2Vec model, extracted the words, and applied **PCA** to reduce the word vectors to 2 dimensions. The code then created a 2D scatter plot, iterating over the first 50 words and annotating each point with its corresponding word.

insights into the suitability of different classifiers for sentiment analysis and other text analytics tasks. The results were as follows:

- Logistic Regression Accuracy: 0.8400
- KNN Accuracy: 0.7815
- Random Forest Accuracy: 0.8122

5 Fine-tuning Pre-trained Language Models (PLMs)

The below models were fine-tuned, colab notebooks containing the fine-tuning code is also linked against each model:

- **DistilBERT:** [Colab Link](#)
- **RoBERTa:** [Colab Link](#)
- **GPT-2:** [Colab Link](#)

We fine-tuned all 3 models using the same set of hyperparameters so that they can be better compared:

1. Batch size = 16 (for DistilBERT and Roberta), 1 for GPT-2
2. Learning Rate = 5e-5
3. Epoch = 3
4. Optimizer = Adam

After Pre-processing the training dataset was split into training and validation set (to monitor loss after each epoch), after which each dataset was encoded using each of the model's tokenizer. A custom PyTorch dataset class was created for the dataset, which pairs tokenized text encodings with their corresponding labels, in order to use them in Pytorch's DataLoader object for batch processing and feeding into models.

For DistilBERT and Roberta fine-tuning was done using native pytorch, in which validation loss was also computed for each epoch. We chose the number of epochs as 3, as they were enough to increase the accuracy of each model from 50% to greater than 90%. For fine-tuning GPT-2 the Trainer API of HuggingFace was utilized.

After fine-tuning we compared each of the model's results with the raw model's performance i.e passing the text from the test data through each of the non fine-tuned models, getting their predictions and computing the accuracy. Performance with and without fine-tuning was similar of all models. We pushed all our fine-tuned models onto huggingface, so that they may be utilized later.

5.1 DistilBERT

Fine tuning of DistilBERT was done on V100 GPU on google colab which took 19.06 minutes. The results with and without fine-tuning of the model are shown below:

DistilBERT - V100 GPU			
Epoch	Training Loss	Validation Loss	Time Taken (s)
1	0.2642	0.2017	383.64
2	0.1457	0.2502	380.19
3	0.0771	0.2307	380.85

Figure 14: DistilBERT Run

DistilBERT performance on Test Dataset		
Metric	Fine-tuned model	Model without Fine-tuning
Accuracy	91.94%	50.33%
Precision	93%	50%
Recall	91%	1%
F1 Score	92%	67%

Figure 15: DistilBERT Performance

5.2 RoBERTa

Fine tuning of Roberta was done on 2 x T4 GPU on Kaggle which took 116 minutes (2 hrs). The model was distributed across both GPUs to get faster results. The results with and without fine-tuning of the model are shown below:

Roberta - 2xT4 GPUs			
Epoch	Training Loss	Validation Loss	Time Taken (s)
1	0.2478	0.2217	2318.24
2	0.1558	0.1844	2322
3	0.1131	0.1833	2321.86

Figure 16: RoBERTa Run

5.3 GPT-2

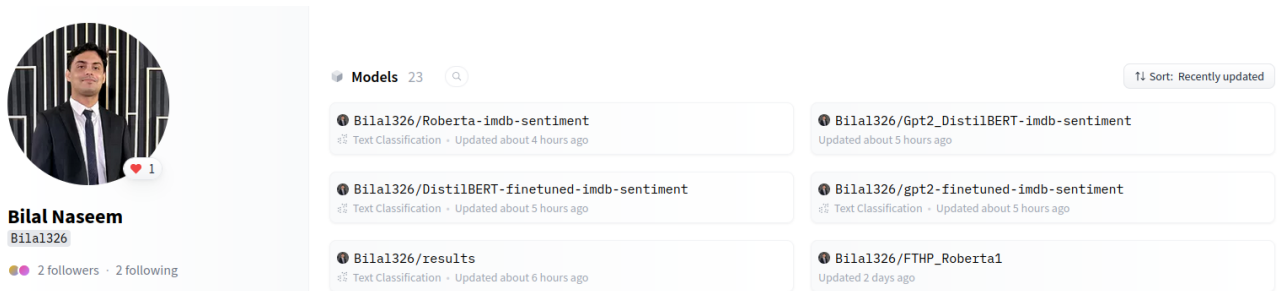
Fine tuning of GPT-2 was done on V100 GPU on google colab which took 118 minutes (2hrs). Floating point was set as 16 to avoid running out of memory. Additionally High-Ram was also enabled for this training. The results with and without fine-tuning of the model are shown below:

Roberta performance on Test Dataset		
Metric	Fine-tuned model	Model without Fine-tuning
Accuracy	93.73%	50.32%
Precision	93%	50%
Recall	94%	1%
F1 Score	94%	67%

Figure 17: RoBERTa Performance

GPT-2 performance on Test Dataset		
Metric	Fine-tuned model	Model without Fine-tuning
Accuracy	93.11%	50.31%
Precision	93%	50%
Recall	94%	1%
F1 Score	93%	67%

Figure 18: GPT-2 Performance



Bilal Naseem
Bilal326
2 followers · 2 following

Models 23

Sort: Recently updated

- Bilal326/Roberta-imdb-sentiment
Text Classification · Updated about 4 hours ago
- Bilal326/Gpt2_DistilBERT-imdb-sentiment
Updated about 5 hours ago
- Bilal326/DistilBERT-finetuned-imdb-sentiment
Text Classification · Updated about 5 hours ago
- Bilal326/gpt2-finetuned-imdb-sentiment
Text Classification · Updated about 5 hours ago
- Bilal326/results
Text Classification · Updated about 6 hours ago
- Bilal326/FTHP_Roberta1
Updated 2 days ago

Figure 19: Models on Huggingface

6 Hybrid Approach

For this section we ensembled our fine-tuned models of section 3 using **Soft Voting**. The code can be found in the following notebook:

- [Colab link](#)

Each of the fine-tuned model was loaded from **huggingface**, and the data was passed through each model and their predictions (**logits**) were recorded. Average predictions were computed using weights and metrics were computed. The results show that accuracy increasing slightly. However it may be increased further by optimizing the weights of the models.

7 Conclusion & Future Work

In section 1, despite using varied approaches and performance metrics, all lexicon-based sentiment analysis tools, including `TextBlob`, `VADER`, and `NLTK Sentiment Intensity Analyzer`, demonstrated comparable sub-optimal accuracy levels on both train and test data sets.

In section 2, we used `TF-IDF` and `Count Vectorization` techniques to represent text data numerically. We found that k-Nearest Neighbours had lesser accuracy and Multinomial Naive Bayes and Random Forest classifiers had higher accuracy. This demonstrates that TF-IDF vectorization and conventional machine learning models are more effective, over `80%`, than lexicon-based techniques in sentiment analysis of IMDB movie reviews.

In section 3, we explored customised word embeddings with an emphasis on Word2Vec models, specifically the Gensim-implemented Skip-gram and Continuous Bag of Words (CBOW) architectures. In order to evaluate the similarity between word vectors and PCA transformation for dimensionality reduction and visualisation, we employed cosine distance measurement. Furthermore, we used the created word embeddings to assess machine learning models, including Random Forest, KNN, and Logistic Regression, on sentiment analysis tasks. The outcomes showed improved accuracy upto `84%`, suggesting that `Word2Vec` embeddings are useful for improving text categorization tasks.

In section 4 we observed that without fine tuning Roberta, DistilBERT, and GPT-2 perform similarly to each other on the IMDB dataset and have accuracy `50%`. After fine-tuning upto 3 Epochs the accuracy quickly jumps greater than `90%`. Ensembling using `Soft Voting` shows accuracy to increase greater than `94%`.

7.1 Future Work

The following points can be tried moving forward in the future on this assignment:

1. Hyperparameter sweep may be performed (using Wandb etc) to get optimal hyperparameters for fine-tuning the models, to get better results.
2. Models may be trained longer for more epochs
3. Other Ensembling methods may be tried out including Stacking. Additionally Techniques of merging models may also be implemented such as `Slerping`, `Task Vector Arithmetic`, `TIES` and `Dare`.