IBA
Introduction to Text Analytics
96110 / 96111

Assignment Title: Sentiment Analysis with Various Approaches

Group Members:

Syed Muhammad Irtiza Zaidi
Muhammad Usman
Azizullah Khan

Contributions:
1. Lexicon-based approach: Azizullah Khan 24931
2. Classical ML approach and word embeddings: Irtiza Zaidi 23973
3. Fine tuning PLMs: Muhammad Usman 24454

Table of Contents

Table of Contents 2
1. Objective 3
2. Dataset 3
3. Tasks 3
3.1. Lexicon-based Approach: 3
Lexicon-based Models Evaluation Results: 3
3.2. Classical Machine Learning Approaches: 4
Findings: 7
3.3 Customized Word Embeddings 7
3.3.1. Word2Vec Embeddings 7
Observations: 9
Conclusion: 9
3.3.2. GloVe Embeddings 10
3.3.3. FASTTEXT 10
3.3.4. GOOGLE WORD2VEC 11
Findings: 11
3.3.5. Paragraph embedding 11
Findings: 12
3.3.6. Custom embedding 13
4. Fine-tuning Pre-trained Language Models (PLMs): 13
Methodology: 13
Results: 13
Hyperparamters of Roberta: 14
Findings: 14
Bibliography 15

1. Objective

The objective of this assignment is to explore and compare various methods for text classification and sentiment analysis, ranging from traditional lexicon-based approaches to modern pre-trained Large Language Models (LLMs).

2. Dataset

Worked with the IMDb movie reviews dataset, containing a large collection of movie reviews labeled as positive or negative sentiment. There are two files: training and testing. We used the training dataset (or portion of it) to train your machine learning model, learn customized Word2Vec or finetune PLMs (RoBERTa, DistillBERT, GPT2). The test data was used for evaluation purposes.

3. Tasks

3.1. Lexicon-based Approach:

In this approach, we conducted sentiment analysis on the IMDb movie reviews test dataset directly using lexicon-based methods. The lexicon libraires used are:

- AFINN
- SentiWordNet
- General Inquirer.
- Opinionfinder
- TEXTBLOB

The analysis focused on measuring the time taken for sentiment predictions with each lexicon and evaluating the models' performance through key metrics, including accuracy, precision, recall, and F1 score. The results offer insights into the efficiency and effectiveness of these lexicon-based approaches for sentiment classification, contributing to a nuanced understanding of their practical applicability in sentiment analysis tasks.

Lexicon-based Models Evaluation Results:

Model Name	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Time**
AFINN	0.85	0.87	0.82	0.84	429.5s
SentiWordNet	0.78	0.75	0.80	0.77	228.5s
General Inquirer	...	...	...	...	Took too much time
TEXTBLOB	0.69	0.63	0.95	0.76	42.2s

Opinionfinder	…	…	…	…	Took too much time

The results shows that AFFIN was the best choice for lexicon approach if we prioritize accuracy however it took the most time among the other models.

3.2. Classical Machine Learning Approaches:

In initial exploration of basic machine learning approaches, a default run was conducted. The process unfolded as follows:

1. Text data preprocessing: This involved cleaning the text data and removing stop words.
2. Vectorization: The preprocessed text was vectorized using default settings of either TF-IDF or CountVectorizer.
3. Model training and evaluation: Several machine learning models were trained and evaluated using the preprocessed and vectorized data.

Model accuracy on the test data is summarized in the table:

Model	Accuracy
Naive Bayes(default)	0.8619
Random Forest(default)	0.8520
KNN (default)	0.7720

The same process was undertaken, but this time **it involved the removal of stop words** in the preprocessing stage:

Model	Accuracy
Naive Bayes(default)	0.8602
Random Forest(default)	0.8406
KNN (default)	0.6851

After incorporating the removal of stop words in the preprocessing stage, the accuracy of default models decreased. However, these initial results provide a baseline understanding and serve as a foundation for further analysis and optimization in the upcoming stages of the project.

In the subsequent training stages, we will adhere to the following preprocessing steps consistently:

1. Preprocessing of text data with the exclusion of stop words.
2. Vectorization of preprocessed text using default settings of TF-IDF or CountVectorizer.
3. Training and evaluation of various machine learning models using the preprocessed and vectorized data.

By following this approach, we aim to improve the accuracy and efficiency of our models in the future.

We then experimented with different variations of machine learning models to improve their fit. The variations were defined for each model:

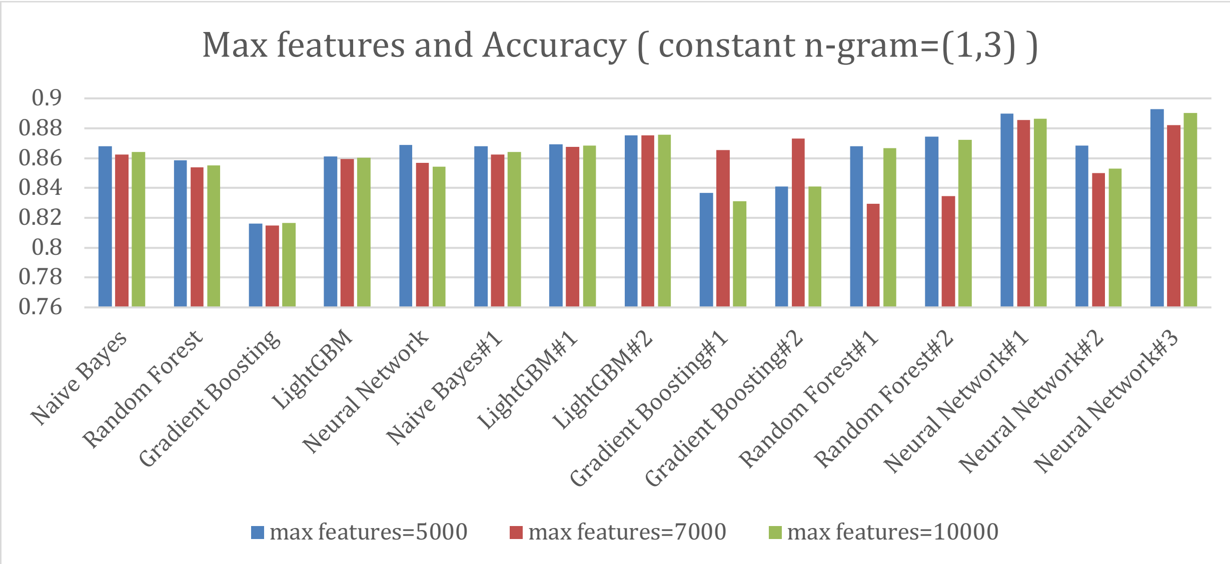
Model	n_estimators	learning_rate	max_depth	min_child_samples	min_samples_split	min_samples_leaf	hidden_layer_sizes	alpha	activation	solver
Naive Bayes										
Random Forest			4		5					
Gradient Boosting	500	0.1	5		5					
LightGBM	1000	0.1	4	4						
Neural Network							100	0.1		adam
Naive Bayes#1										
LightGBM#1	500	0.1	5	5						
LightGBM#2	1000	0.1	4	4						
Gradient Boosting#1	500	0.1	5		5					
Gradient Boosting#2	1000	0.1	4		5					
Random Forest#1	500		4		5					
Random Forest#2	1000		7		4	2				
Neural Network#1							100	0.1		adam
Neural Network#2							100,50	0.1	tanh	adam
Neural Network#3							50,50	0.1	logistic	adam

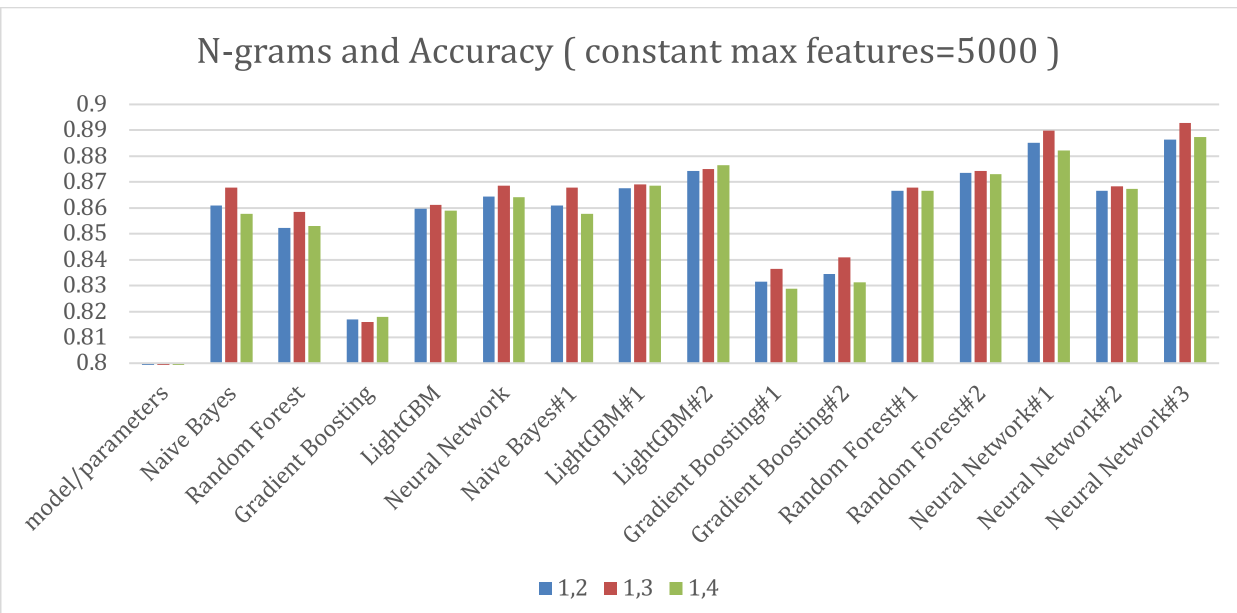
Utilizing the same default preprocessing and vectorization procedures, we executed the modified machine learning models. The resulting benchmarks were as follows:

model/parameters	Default		Ngram Range: (1, 2) Maximum Features:5000 Sublinear Term Frequency: True		Ngram Range: (1, 3) Maximum Features:5000 Smooth Inverse Document Frequency:		Ngram Range: (1, 3) Maximum Features:7000 Sublinear Term Frequency: True		Ngram Range: (1, 3) Maximum Features:10000 Smooth Inverse		Ngram Range: (1, 4) Maximum Features:5000 Smooth Inverse Document Frequency: True	
Model	Accuracy	Training Time (s)	Accuracy	Training Time (s)	Accuracy	Training Time	Accuracy	Training Time (s)	Accuracy	Training Time	Accuracy	Training Time (s)
Naive Bayes	0.86185	0.10321	0.86095	0.05122	0.86795	0.10463	0.8626	0.04535	0.86405	0.04895	0.85765	0.05026
Random Forest	0.8538	106.23125	0.8524	74.08296	0.85845	91.19682	0.8538	73.99569	0.85495	71.32173	0.85295	91.24178
Gradient Boosting	0.81445	140.13324	0.817	106.32915	0.816	141.76691	0.81475	106.22161	0.8166	115.45152	0.81785	Not Available
LightGBM	0.85495	55.96297	0.85965	20.28297	0.8613	82.31538	0.8592	38.81535	0.86045	30.20604	0.859	50.00488
Neural Network	0.88225	514.89441	0.8644	169.01804	0.86865	319.24255	0.85675	165.03389	0.8543	206.61027	0.86405	210.66834
Naive Bayes#1	0.86185	0.06411	0.86095	0.04876	0.86795	0.05406	0.8626	0.08536	0.86405	0.04808	0.85765	0.04535
LightGBM#1	0.8647	123.43748	0.86765	33.68337	0.8691	135.09792	0.8676	63.91763	0.8683	48.07481	0.8685	80.6095
LightGBM#2	0.87215	180.75068	0.8743	44.9809	0.87505	191.66809	0.8751	84.41437	0.8756	62.66392	0.8764	108.54124
Gradient Boosting#1	0.8608	1181.17172	0.8315	9.90384	0.8365	8.93834	0.86555	871.42061	0.8313	8.45947	0.8289	10.95057
Gradient Boosting#2	0.86805	1810.2214	0.8345	33.75749	0.841	29.04692	0.8732	1643.47058	0.841	28.46131	0.83125	37.92568
Random Forest#1	0.84185	5.82299	0.86665	878.26025	0.86785	1173.61309	0.8296	10.1878	0.86675	969.77331	0.8666	1000.09142
Random Forest#2	0.8469	20.11162	0.8736	1403.65374	0.8742	1856.63098	0.83435	34.6264	0.8723	1545.34157	0.87315	1570.63248
Neural Network#1	0.87475	2402.30288	0.88515	70.57317	0.8899	208.6413	0.8855	56.67385	0.8864	107.68243	0.8822	55.60122
Neural Network#2	0.87315	5078.92767	0.86655	111.36222	0.86845	379.45617	0.85	302.53694	0.85295	127.79568	0.86745	193.94654
Neural Network#3	0.8917	2797.29417	0.88635	133.20806	0.89275	882.68432	0.88195	279.20213	0.8902	297.96224	0.8874	168.17284
Maximum	0.8917	5078.92767	0.88635	1403.65374	0.89275	1856.63098	0.8855	1643.47058	0.8902	1545.34157	0.8874	1570.63248
Minimum	0.81445	0.06411	0.817	0.04876	0.816	0.05406	0.81475	0.04535	0.8166	0.04808	0.81785	0.04535

(The values have been rounded to 5 decimal places)

Based on the benchmark results and data of all models, here are some insightful visualizations:





These visualizations provide a clear understanding of the performance and accuracy of each model, allowing for easier comparison and analysis. They can also help identify patterns and trends in the data, aiding in the selection of the most suitable model for the task at hand.

The bar graphs above illustrate the relationship between maximum features and accuracy, with the n-gram range fixed at (1,3). Additionally, another set of bar graphs depicts the relationship between n-grams and accuracy, while maintaining a constant maximum feature value of 5000.

Findings:

- Neural networks emerged as top performers among classical models, showcasing superior accuracy despite longer training times.
- Naïve Bayes, with a faster training pace, still delivered commendable accuracy.
- Gradient Boosting offered a balanced trade-off between accuracy and training time, outperforming Naïve Bayes but requiring more training time.

The optimal choice depends on specific task requirements, considering factors like computational resources, time constraints, and desired accuracy levels.

3.3 Customized Word Embeddings

In this section, we implemented a traditional approach with a unique modification - the incorporation of customized word embeddings. Word embeddings provide dense representations of words in a continuous vector space, capturing complex semantic relationships.

3.3.1. Word2Vec Embeddings

Word embeddings provide dense representations of words within a continuous vector space, encapsulating intricate semantic relationships.

Here are the steps we followed:

1. Utilize the Word2Vec model from the Gensim library.
2. Create a custom function to calculate the average word vectors for each review, based on the Word2Vec model.
3. Use the word embeddings generated by Word2Vec as feature representations for both the training and test sets.
4. Train various machine learning models using these word embeddings.
5. Evaluate the performance of each model and store them in a models dictionary.
6. Calculate and display key performance metrics for each model.

The following machine learning models were employed in these variations:

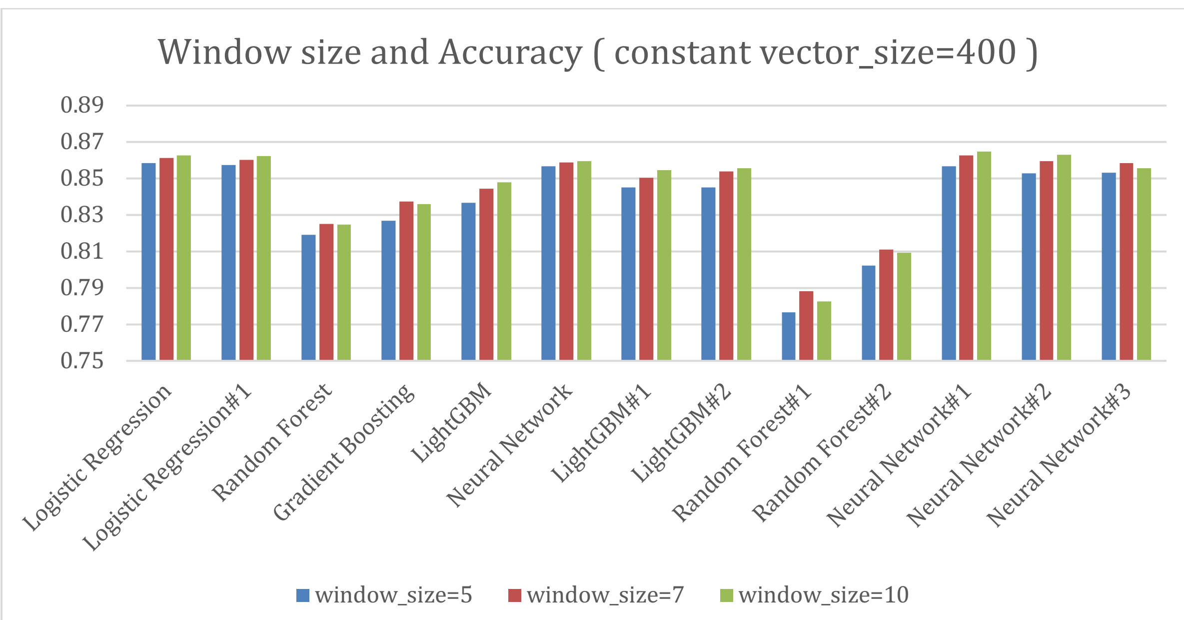
Model	solver	n_estimators	learning_rate	max_depth	min_child_samples	min_samples_split	min_samples_leaf	hidden_layer_sizes	alpha	activation
Logistic Regression	Default									
Logistic Regression#1	'lbfgs'									
Random Forest	Default									
Gradient Boosting	Default									
LightGBM	Default									
Neural Network	'adam'							25	0.1	
LightGBM#1		500	0.1	5	5					
LightGBM#2		1000	0.1	4	4					
Random Forest#1		500		4		5				
Random Forest#2		1000		7		4	2			
Gradient Boosting#1		500	0.1	5		5				
Gradient Boosting#2		1000	0.1	4		5				
Neural Network#1	'adam'							25	0.1	
Neural Network#2	'adam'							50,25	0.1	tanh
Neural Network#3	'adam'							50,50	0.1	logistic

***If a model is not included in the result table, it indicates that the training process for that model exceeded a duration of two hours and was consequently canceled.**

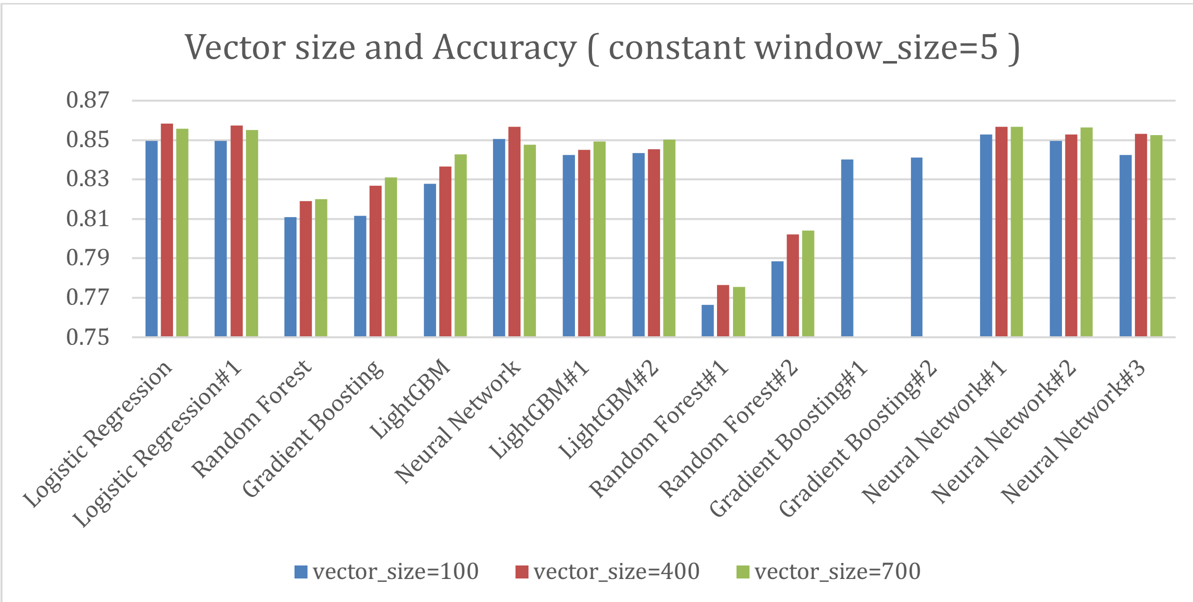
model/parameters	Vector Size: 100 Window Size: 5 Minimum Word Count: 1 Number of workers : 4		Vector Size: 100 Window Size: 7 Minimum Word Count: 1 Number of workers : 4		Vector Size: 400 Window Size: 5 Minimum Word Count: 1 Number of workers : 4		Vector Size: 400 Window Size: 7 Minimum Word Count: 1 Number of workers : 4		Vector Size: 400 Window Size: 10 Minimum Word Count: 1 Number of workers : 4		Vector Size: 700 Window Size: 5 Minimum Word Count: 1 Number of workers : 4	
Model	**Accuracy**	**Training Time**	**Accuracy**	**Training_Time**	**Accuracy**	**Training Time**	**Accuracy**	**Training Time**	**Accuracy**	**Training Time**	**Accuracy**	**Training_Time**
Logistic Regression	0.84945	4.34616	0.85005	4.13501	0.8584	19.1542	0.86135	17.37677	0.8626	19.25628	0.8557	32.49012
Logistic Regression#1	0.84965	0.57226	0.85045	0.72289	0.8573	1.22185	0.86015	1.9281	0.86245	2.1801	0.85515	1.92499
Random Forest	0.81105	31.28532	0.8112	38.39308	0.81905	63.03436	0.825	77.67354	0.8246	82.35201	0.8202	83.15036
Gradient Boosting	0.81155	110.38702	0.8122	134.63199	0.82695	449.79385	0.8374	570.54636	0.83585	597.03796	0.831	782.09666
LightGBM	0.82795	3.1633	0.8299	3.80029	0.8366	12.57444	0.8444	16.90488	0.84795	19.29697	0.8427	22.95572
Neural Network	0.8506	95.33654	0.8518	71.55212	0.85665	235.92953	0.8588	246.91646	0.85935	256.05625	0.8478	449.06059
LightGBM#1	0.8426	8.79742	0.84095	10.60129	0.845	35.18515	0.85045	46.99048	0.8546	50.53566	0.8493	64.5583
LightGBM#2	0.8434	13.38678	0.84505	14.31238	0.84525	55.87564	0.85395	60.21709	0.85575	66.31137	0.8503	93.29426
Random Forest#1	0.7663	52.63709	0.76145	62.96355	0.77665	111.37776	0.78825	129.10137	0.78245	137.3025	0.7755	135.16774
Random Forest#2	0.7886	167.48554	0.78815	202.14406	0.80225	331.57042	0.8109	421.1015	0.80945	441.66335	0.80415	433.74606
Gradient Boosting#1	0.8403	900.2742	0.8396	1094.45106	**Took too much time**							
Gradient Boosting#2	0.84115	1478.91855	0.84285	1811.22976								
Neural Network#1	0.8527	16.65883	0.85445	27.15138	0.85675	15.42282	0.8626	31.44824	0.8648	60.57322	0.8568	26.47466
Neural Network#2	0.84945	14.98636	0.8524	42.55616	0.8529	17.92619	0.85935	35.79421	0.86295	55.45148	0.85635	52.35299
Neural Network#3	0.8425	21.31429	0.84555	19.74318	0.8533	30.91832	0.85845	49.60375	0.85575	60.40345	0.8524	30.28563
Maximum	0.8527	1478.91855	0.85445	1811.22976	0.8584	449.79385	0.8626	570.54636	0.8648	597.03796	0.8568	782.09666
Minimum	0.7663	0.57226	0.76145	0.72289	0.77665	1.22185	0.78825	1.9281	0.78245	2.1801	0.7755	1.92499

Here are two visualizations:

Visualization 1: Changing window size and keeping vector size constant.



Visualization 2: Keeping window size constant and changing vector size.



Observations:

- In the word2vec context, the neural network emerged as the top-performing model, closely followed by logistic regression, which required significantly less training time.
- Random forest and gradient boost models exhibited longer training times with less satisfactory results.
- Logistic regression proved to be a balanced choice in terms of efficiency and performance, while neural networks stood out as the top-performing model.
- Increasing vector size did not consistently improve accuracy, and in fact, accuracy decreased when the vector size was increased from 400 to 700.
- Increasing the window size from 5 to 7 consistently led to improved accuracy across all cases, although this improvement is not guaranteed in every scenario.

Conclusion:

Multiple trials are necessary to determine the optimal parameters for the task at hand.

3.3.2. GloVe Embeddings

GloVe, short for Global Vectors for Word Representation, is an unsupervised learning algorithm aimed at producing word embeddings.
In our current task, we leverage GloVe embeddings to enhance sentiment analysis. GloVe's strength lies in its capacity to capture global semantic relationships in word representations, making it an ideal choice for improving sentiment comprehension in the text data utilized for analysis. By employing GloVe embeddings, we aim to achieve more accurate and nuanced sentiment analysis results.

model/parameters	100-dimensional GloVe model		200-dimensional GloVe model		300-dimensional GloVe model	
Model	Accuracy	Training_Time	Accuracy	Training_Time	Accuracy	Training_Time
Logistic_Regression	0.76055	1.0528	0.7892	5.43461	0.7998	8.44435
Logistic_Regression#1	0.7605	0.63645	0.78885	0.82152	0.7998	1.15702
Random_Forest	0.7231	32.38623	0.7401	45.01067	0.73665	54.0396
Gradient_Boosting	0.73015	111.31173	0.75475	222.51448	0.75225	335.94614
LightGBM	0.74565	3.21289	0.76805	6.31619	0.77305	9.83011
Neural_Network	0.75565	139.01023	0.79305	164.69155	0.80395	208.38793
LightGBM#1	0.75605	9.18829	0.78615	18.26381	0.7904	28.3527
LightGBM#2	0.7593	13.29563	0.78555	27.07395	0.79415	41.26826
Random_Forest#1	0.6816	52.55409	0.69145	73.3228	0.6899	89.26775
Random_Forest#2	0.70095	168.58667	0.716	236.16839	0.7177	287.11298
Neural_Network#1	0.76605	20.928	0.79175	36.15513	0.80515	37.1512
Neural_Network#2	0.7622	7.86412	0.7892	16.83604	0.79895	26.7379
Neural_Network#3	0.7599	42.64506	0.7858	39.96212	0.79835	34.2797
Maximum	0.76605	168.58667	0.79305	236.16839	0.80515	335.94614
Minimum	0.6816	0.63645	0.69145	0.82152	0.6899	1.15702

3.3.3. FASTTEXT

Fasttext is an open-source, free, lightweight library that allows users to learn text representations and perform text classification tasks. It was developed by Facebook's AI Research (FAIR) lab.

Model	Accuracy	Training_Time
Logistic_Regression	0.7866	2.38199
Logistic_Regression#1	0.7866	2.81116
Random_Forest	0.75145	70.24767
Gradient_Boosting	0.7619	447.75446
LightGBM	0.7812	14.79796
Neural_Network	0.83115	525.14708
LightGBM#1	0.8048	38.66403
LightGBM#2	0.81135	51.00005
Random_Forest#1	0.71355	115.59462
Random_Forest#2	0.73925	372.70524
Neural_Network#1	0.8167	18.97893
Neural_Network#2	0.8177	44.81206
Neural_Network#3	0.49675	14.78099
Maximum	0.83115	525.14708
Minimum	0.49675	2.38199

3.3.4. GOOGLE WORD2VEC

Google's Word2Vec is a renowned word embedding model introduced by a team of researchers at Google, led by Tomas Mikolov, in 2013. The primary objective of Word2Vec is to represent words as continuous vector spaces, effectively capturing semantic relationships and similarities between them. Here are some notable features of Google's Word2Vec:

Model	Accuracy	Training_Time
Logistic Regression	0.8181	3.58211
Logistic Regression#1	0.81795	3.4727
Random Forest	0.76605	67.6552
Gradient Boosting	0.7773	449.13656
LightGBM	0.7953	15.0785
Neural Network	0.8222	325.74254
LightGBM#1	0.81355	39.3996
LightGBM#2	0.8194	51.45763
Random Forest#1	0.7285	116.60547
Random Forest#2	0.7515	375.35187
Neural Network#1	0.81735	42.71166
Neural Network#2	0.82405	30.23562
Neural Network#3	0.8166	57.20507
Maximum	0.82405	449.13656
Minumim	0.7285	3.4727

Findings:

Across Glove, FastText, and GoogleWord2Vec embedding approaches, overall model performance decreased. Neural networks consistently outperformed other models, with logistic regression remaining a competitive choice, characterized by shorter training times.

3.3.5. Paragraph embedding

Paragraph embeddings involve representing entire paragraphs or documents as continuous vectors, capturing their overall semantic meaning. Unlike word embeddings that focus on individual words, paragraph embeddings provide a holistic understanding of the text. These embeddings are particularly useful in various natural language processing tasks, such as text classification, document clustering, and information retrieval.
Here are the steps to generate paragraph embeddings:

Steps:

1. Define Functions:
    - Create a function to calculate the average word vectors.
    - Develop a function to calculate the paragraph embeddings.
2. Preprocess Reviews:
    - Clean and preprocess the training and test reviews by removing stop words, punctuation, and applying stemming techniques.
3. Train Word2Vec Model:
    - Train a Word2Vec model on the preprocessed reviews.
4. Combine Features:
    - Concatenate or average the word vectors and paragraph embeddings for each review to create a comprehensive feature set.
5. Result:
    - Utilize the combined features for various NLP tasks, such as text classification or document clustering, to achieve improved performance.

model/paramters	vector_size: 200 window: 5 min_count: 1 workers: 4		vector_size: 300 window: 7 min_count: 1 workers: 4		vector_size: 100 window: 5 min_count: 1 workers: 4		vector_size: 200 window: 7 min_count: 1 workers: 4	
Model	**Accuracy**	**Training_Time**	**Accuracy**	**Training_Time**	**Accuracy**	**Training_Time**	**Accuracy**	**Training_Time**
Logistic_Regression	0.85885	16.69147	0.86255	28.67242	0.8486	9.16007	0.85845	19.15381
Logistic_Regression#1	0.85685	2.1948	0.86255	1.72472	0.84825	1.26366	0.85815	1.26582
Random_Forest	0.81705	80.93265	0.8198	79.2486	0.80915	54.17364	0.82	65.39479
Gradient_Boosting	0.82075	591.4976	0.82725	715.5936	0.81155	309.05073	0.82785	463.78666
LightGBM	0.8344	19.5023	0.8414	19.44444	0.82545	9.97484	0.8389	13.16874
Neural_Network	0.8578	127.14026	0.857	163.89313	0.8485	74.30302	0.84765	121.89225
LightGBM#1	0.84805	50.05583	0.85125	57.49448	0.83735	23.8271	0.8467	37.40793
LightGBM#2	0.84725	64.13657	0.8534	81.96592	0.8377	30.3333	0.84885	54.9498
Random_Forest#1	0.7636	134.43644	0.78435	130.98669	0.75285	97.04185	0.7785	107.75469
Random_Forest#2	0.7949	433.27276	0.80605	417.95796	0.78615	306.09759	0.8005	347.59858
Neural_Network#1	0.8564	178.06416	0.8623	75.54604	0.85185	53.72391	0.85765	25.57386
Neural_Network#2	0.8544	25.9078	0.86055	28.41462	0.8494	33.42154	0.852	39.87366
Neural_Network#3	0.85485	26.33476	0.8467	36.94417	0.84705	26.51324	0.8535	31.67174
Maximum	0.85885	591.4976	0.86255	715.5936	0.85185	309.05073	0.85845	463.78666
Minimum	0.7636	2.1948	0.78435	1.72472	0.75285	1.26366	0.7785	1.26582

Findings:

Based on our analysis, the naïve bayes algorithm emerged as the clear winner, offering not only superior accuracy but also requiring less time for training and implementation. This efficiency and effectiveness make it an ideal solution for our needs.

Furthermore, we observed that incorporating paragraph embeddings, particularly with a vector size of 100 and a window of 5, led to improved accuracy compared to scenarios where paragraph embeddings were not utilized. This highlights the potential benefits of using paragraph embeddings in enhancing the performance of our model.

3.3.6. Custom embedding

During the document classification task, an attempt was made to create custom embeddings for the training documents. However, the process proved to be computationally intensive and required an extensive amount of time for training the models on the given data. As a result, the process had to be terminated without obtaining the desired custom embeddings.

4. Fine-tuning Pre-trained Language Models (PLMs):

In this section, we detail the process of fine-tuning three different PLMs – DistilBERT, RoBERTa, and GPT-2 – for sentiment analysis on the IMDb movie reviews dataset. We describe the methodology, the hyperparameters of the winning model, and comparative analysis of each PLM-based approach.

Methodology:

All of the fine tuning is done through pytorch and transformers library.

1. Load the datasets into the notebook.
2. Initialize the model and tokenizer for that model.
3. Define the function for computing accuracy of the model.
4. Tokenize the datasets.
5. Add labels of 1 and 0 to the sentiment column corresponding to 'positive' and 'negative' respectively.
6. Define the hyperparameters in the TrainingArguments().
7. Define the model, training arguments, labelled datasets, and metric function in the Trainer().
8. Train the model and print its accuracy and training time.

Results:

Model	Accuracy	Training time/s	Parameters
Distilbert	93.2%	3176.83853	66M
Roberta	94.9%	6044.41686	125M
GPT2	93.3%	5775.63663	117M
Distilbert (benchmark)	83.3%	-	66M

Hyperparamters of Roberta:

1. num_train_epochs=3
2. per_device_train_batch_size=8
3. per_device_eval_batch_size=8

Findings:

1. Even though Roberta is the winning PLM, it is worth noting that its training time is almost twice as much as Distilbert, with only around 2% increase in the accuracy. Since all of the three PLMs were trained on the same hyperparameters, the only reason for Roberta's slightly better performance is its larger memory footprint. Therefore, using Distilbert for this task is a much better trade off.

2. The reason Distilbert outperformed distilbert-base-uncased-finetuned-sst-2-English is probably because of the hyperparameter tuning. The benchmark version uses a slower learning rate compared to mine which might have caused the model to converge slowly or get stuck in a local minima. Plus, my model uses smaller batches which may have helped it to generalize better and prevent overfitting. In short, my model learned more relevant patterns and features, leading to improved accuracy.

Bibliography

All Jupyter Notebooks are available at this location:
https://drive.google.com/drive/u/0/folders/1ELc0Q0RPNmHiKK7WjSDohkxEiMxeTVl-