TEXT ANALYTICS - 96944

INSTRUCTOR: DR. SAJJAD HAIDER

IMDb Sentiment Analysis
Assignment # 3

Group members:
Abdul Haseeb (17132)
Annayah Usman (18673)
Sawera Hanif (29413)

Date: January 8, 2025

Abstract

This project is a methodological and comprehensive empirical comparative analysis of employing various approaches for sentiment analysis on the popular IMDb dataset. The main objective and theme is Finetuning PLMs using LoRA and PEFT while also using classical ML models for the same purpose. The baseline is also the famous, known already finetuned PLM, distilbert-base-uncased-finetuned-sst-2-english. The approach of this study is to train and test 5 classical ML models, which include Naive Bayes, Logisitic Regression, k-NN, Random Forest and Gradient Boosting, with first a TFIDF vectorizer solely and then combining a Word2Vec embedding. Several different configurations and combinations of hyperparameters are checked to asses the results. Next, for finetuning, 4 PLMs: DistilBERT, RoBERTa, ALBERT and GPT2 are explored using LoRA configurations and PEFT. They are experimented in two setups for portion of the training and test datasets, first by fixing LoRA configurations to reasonable values and changing training hyperparameters such as batch size, learning rate and number of epochs. In the second set, the training hyperparameters are logically fixed and LoRA configurations are changed. This creates a holistic set of combinations by which losses, metrics, % of trainable parameters and training times are observed. Then final configurations and hyperparameter values are decided and set for each PLMs which results in final finetuning on the complete datas. This study concludes after pushing all of our finetuned models to HuggingFace. Baseline finetuned model and Classical ML section was done on Google Colab T4 GPU, while all PLM finetuning work was done on Kaggle GPU T4 x2.

Contribution Statement:

1. **Annayah Usman:** Classical ML models, Finetuning experimentations, Final finetuning
2. **Abdul Haseeb:** Classical ML models, Finetuning experimentations
3. **Sawera Hanif:** EDA, finetuning experimentations

Link to Github repo: GitHub repo for this project

Contents

1 Introduction

1.1 About the IMDb dataset

The IMDb sentiment analysis dataset is widely used in natural language processing (NLP) tasks. It consists of user-generated movie reviews labeled with sentiments (positive or negative) indicating the moviegoers experience and the feelings the movie elicited in them.

1.2 Exploratory Data Analysis

1.2.1 Sentiment Distribution

The visual in Figure 1 shows an almost equal distribution of positive and negative sentiments in the train and test datasets, with approximatley 12,500 reviews per class. This balance ensures no bias during training, promoting better model performance across both sentiment types.

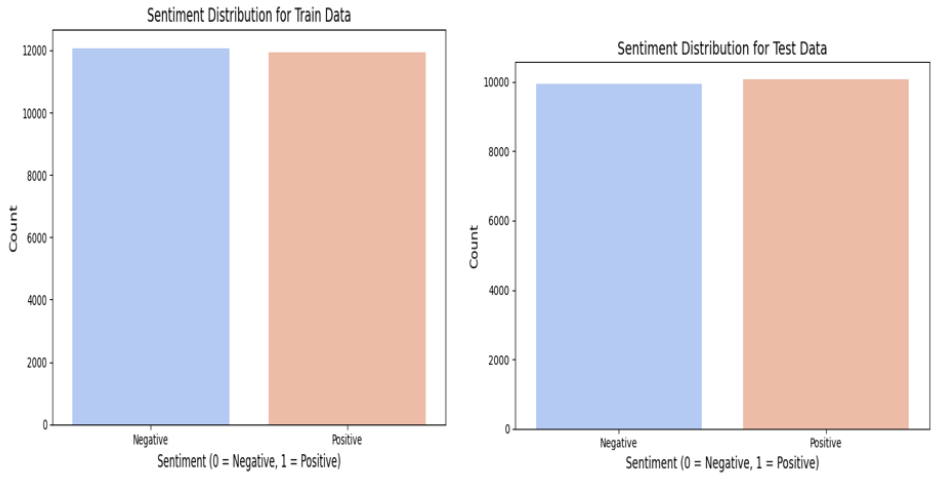

Figure 1: Distribution of sentiments

1.2.2 Word Cloud Analysis

Figure 2 shows separate word clouds for positive and negative sentiments in both train and test datasets highlighting the frequent terms occurring in both datasets. In positive reviews, Words like ”movie”, ”film”, ”character”, ”love”, ”good”, ”people”, and ”one” appear prominently, reflecting their relevance in them. While words like ”film”, ”movie”, ”bad”, and ”even” dominate, appear as commonly used terms in negative feedback.

Figure 2: Word Cloud of Train Data

1.2.3 Token Frequency Distribution

In Figure 3 The distribution of top unigrams for positive and negative reviews indicates the common tokens contributing to sentiment classification.

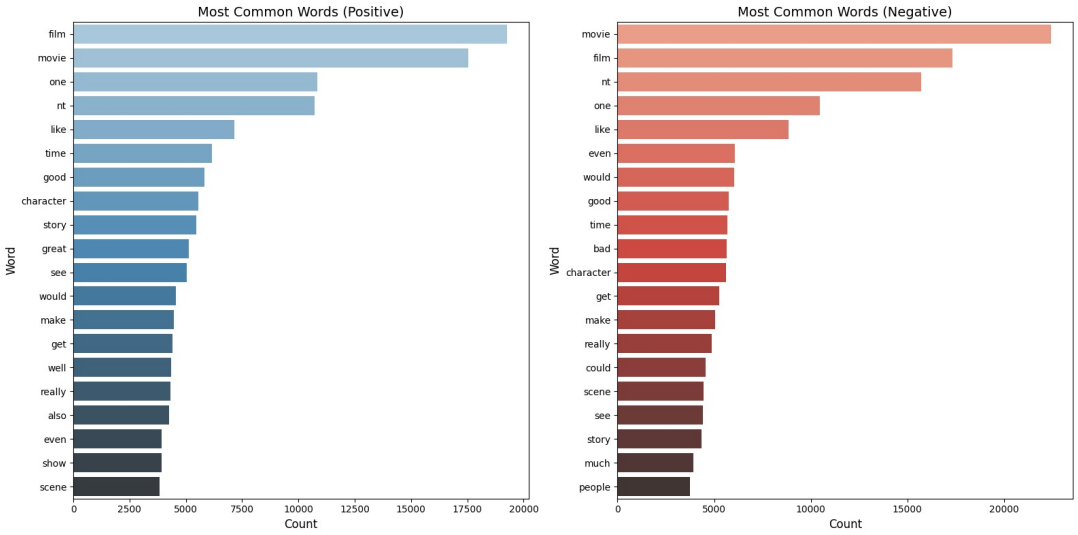

Figure 3: Distibution of Token Frequencies

1.2.4 Review Lengths Distribution

Figure 4 show the review length distribution for positive and negative sentiments in both datasets. Reviews vary widely in length, but negative reviews tend to have slightly longer average lengths.

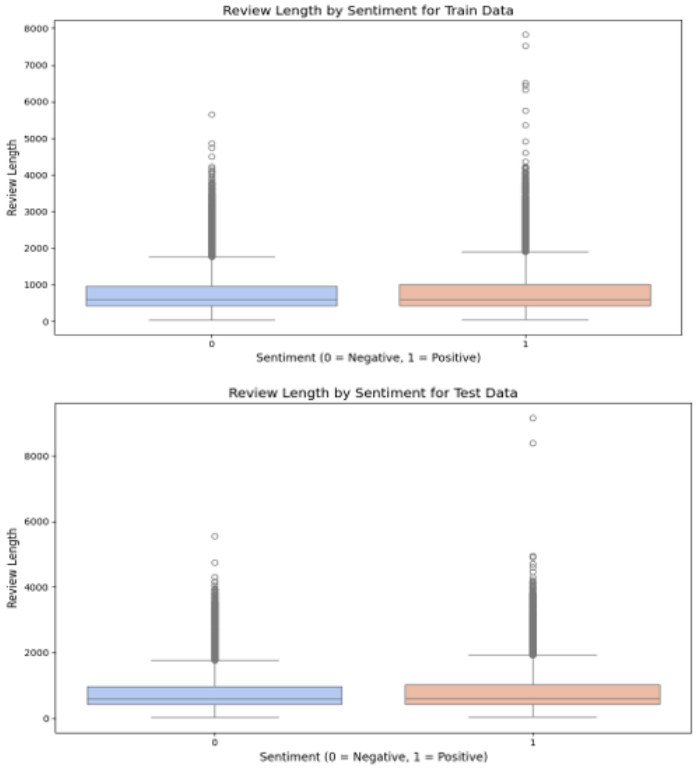

Figure 4: Distibution of Review Lengths

2 Baseline Finetuned Model

The baseline model was the distilbert-base-uncased-finetuned-sst-2-english. For the purpose of comparison, this model was tested on the complete testing dataset. The overall accuracy stood as 0.8904, precision at 0.9163, recall at 0.8610 and F1 score at 0.8878. The detailed classification report can be found in Table 1:

Class	Accuracy	Precision	Recall	F1-Score	Support
negative	0.89	0.87	0.92	0.89	9935
positive	0.89	0.92	0.86	0.89	10065
Weighted Avg	0.89	0.89	0.89	0.89	20000

Table 1: Baseline Classification Report on Test Data

When testing our baseline model, we tried it first by passing the raw test data, uncleaned, at it is. The second approach was to preprocess/clean that data for just the HTML tags and special-like characters, not too much strenuous preprocessing and then test the baseline. We found that the second approach did improve our metrics but only slightly and not as significantly from having passed the raw test data; the accuracy and F1 score stay at 0.89. To keep our experiments and comparisons consistent, the reported results in Table 1 are from the second approach of having passed slightly processed reviews to the baseline.

3 Classical ML Models

The methodology followed for this section entailed a custom and thorough preprocessing function, employing the TFIDF vectorizer but in two different ways: one using different combinations of hyperparameters of TFIDF (ngram_range and max_features) solely and the second was combining it with a Word2Vec embedding. The ML models used were Naive Bayes, Logistic Regression, Random Forest, Gradient Boosting and k-NN.

3.1 Preprocessing

Using the NLTK and re supporting libraries, a nested preprocessing function was created. It performed removal of punctuation, html tags, urls, special characters, then converting to lowercase, afterwhich removing stopwords and then lemmatization of the text. This preprocessing was used consistently throughtout all our ML models experimentations. On the other hand, the sentiments were mapped from negative and positive to 0 and 1, respectively. This mapping stays consistent throughout all other experimentations of finetuning as well.

3.2 TFIDF Vectorization with Different Configurations

The first approach to vectorization was to train, validate and then test solely through the TFIDF vectorizer using various combinations of the hyperparameters of the vectorizer: ngram_range and max_features. Please note that the each of the models' hyperparameters were used as default in these experimentations; we instead chose to explore with the vectorizer itself.

This experimentation was evaluated on the complete test data based on the accuracy and they are reported in the table in Figure 5.

Model\Config	Unigram and No Cap on Max Features	Uni- & Bi-gram and Max Features=100	Uni- & Bi-gram and Max Features=200	Uni- & Bi-gram and Max Features=500	Uni- & Bi-gram and Max Features=2000	Uni- & Bi-gram and Max Features=5000	Uni-, Bi- & Tri-gram and Max Features=5000	Uni- till 5-gram and Max Features=2000
Naive Bayes	0.86	0.73	0.76	0.82	0.85	0.86	0.86	0.85
Logistic Regression	0.89	0.73	0.78	0.84	0.88	0.88	0.88	0.88
k-NN	0.76	0.65	0.67	0.69	0.70	0.73	0.73	0.70
Random Forest	0.85	0.72	0.76	0.81	0.84	0.85	0.85	0.84
Gradient Boosting	0.81	0.72	0.75	0.80	0.81	0.81	0.81	0.81

Figure 5: Test Accuracies of TFIDF Configurations across ML Models

We observe that increasing the diversity in the n-grams increases accuracy across all models. However, the cap on max features may tend to limit or keep the accuracies consistent even with a greater collection on words by the n-grams. This suggests that is the n-grams were increased and there was no cap on the max_features, we may expect improved performance on the better performing models from the group.

We find that k-NN performs the poorest of all the models with accuracies relatively much lower than the rest. While logistic regression performs the best consistently yielding the highest accuracy of 0.89 which is the same as the baseline. But overall, most of the classical models in this experimentation perform below average to the standard set by the baseline. The execution of this experimentation took about 45 minutes.

3.3 TFIDF Vectorization with Word Embedding

The idea for this experimentation was to combine a word embedding with the TFIDF vectorization. For this we used the Word2Vec embedding of the Gensim library. We trained the embedding which had a dimension of 300 and window of 5. The embedding was then obtained and stacked with the TFIDF vectorized matrix to get a slightly more dense and rich matrix. Our models, except for Naive Bayes due to the induction of negative values through the embedding, were tested the same way but now with limited TFIDF configurations. The results of which can be found in the table of Figure 6.

Model\Config	Unigram and No Cap on Max Features	Uni- & Bi-gram and Max Features=2000	Uni-, Bi- & Tri-gram and Max Features=5000
Logistic Regression	0.89	0.88	0.89
k-NN	0.79	0.78	0.79
Random Forest	0.81	0.83	0.82
Gradient Boosting	0.85	0.85	0.85

Figure 6: Testing Accuracies of TFIDF with Gensim Word2Vec Embedding

Surprisingly, the accuracies don't improve significantly for the models that were performing well. For e.g., Logistic still plateaus at 0.88-0.89. While though k-NN improves, it is still the poorer performing models in the group. This could be due to the way we combined the vectorized matrix and word embedding together.

The above approach took over 1 hour for execution. While we also went ahead and downloaded a pre-trained word embedding word2vec_model.bin, we were not able to work on it further. But this second approach was also interesting to explore where it was planned to weight the pre-trained embedding using the TFIDF. Apply PCA for dimensionality reduction and then proceed with model training and evaluations.

Nonetheless, the current approach of the word embedding along with TFIDF vectorization does not outperform our baseline in general again. Only one classical ML model performs at par to the finetuned baseline.

4 Finetuning experimentations using PLMs

For finetuning with LoRA, we explored 4 PLMs: DistilBERT, RoBERTa, ALBERT and GPT2. Since the objective of this project is to make a comparative analysis of the results by changing training hyperparameters and/or LoRA configurations, the methodology for experimentation in this section are outlined as exactly so:

Once we had loaded the base PLM(s) and tokenized our reviews by the respective PLM tokenizer, we ran a first set of experiments, Experiments Phase 1, by keeping the LoRA configurations logically fixed and changed training hyperparameters such as batch size, number of epochs and learning rate. Therefore, we generated a number of combinations within this setup to assess our finetuning performance by observation of training and validation losses and the evaluation metrics. The test dataset results would then be reported for each combination. While in the second set of experiments, Experiments Phase 2, we fixed the values of training hyperparameters of batch size, etc. and changed the LoRA configurations such as: for target matrices first either finetuning the Query matrix, then finetuning the Query matrix along with the Key matrix and then finetuning both with the Value matrix, etc.

This pipeline of experimentation was applied to all 4 PLMs, and in this report we shall report the metrics of accuracy, precision, recall and F1 score on the test data for comparison to baseline and ML models.

It must also be noted that in order to get an idea of which hyperparameters and configurations for LoRA the model should be fine tuned for, just for experimentation purposed we used 10% of both the training and test data keeping random state as 42. Once the configurations were determined from the experimentations, after we eyeballed and logically observed the combinations' effects on the classification, we used them in application to our final fine tuning which was done on the complete training and testing dataset for all PLMs, with the exception of GPT2 for which we used 5% of both datas for final finetuning.

Please also note that in a few cases, some configurations may be smaller than others. For example, some models may have been experimented for batch sizes 4 and 8 while some for 8 and 16. This is purely to avoid memory issues and constraints on time and thus have the PLMs also run smoothly. To further account for this, for PLMs such as RoBERTa we had to explicitly state fp16=True in the training arguments so that it does mixed precision training and avoids issues of going out of memory as this was experienced in the initial cycles of running the pipeline.

4.1 DistilBERT

For Experiments Phase 1 the configurations, fixed and changeable can be found in Figure 7:

```
# Fixed LoRA parameters
rank = 8
target_matrices = ["attention.q_lin", "attention.k_lin", "attention.v_lin"]
lora_alpha = 16
lora_dropout = 0.1

# Changing hyperparams for batch size, epochs and learning rates
batch_sizes = [16, 32]
epochs_list = [3, 5]
learning_rates = [3e-5, 1e-4]

training_dropout = 0.1 # Fixed
```

Figure 7: DistilBERT Experiments Phase 1 Configurations

For the Experiments Phase 2, the configurations can be found in Figure 8:

```
# Fixed parameters for batch size and epochs, etc
fixed_batch_size = 16
fixed_epochs = 5
fixed_learning_rate = 1e-4
training_dropout = 0.1

# LoRA parameter combinations
ranks = [8, 16]
target_matrices_list = [["attention.q_lin"], ["attention.q_lin", "attention.k_lin"], ["attention.q_lin", "attention.k_lin", "attention.v_lin"]]
lora_alpha = 16
lora_dropouts = [0.1, 0.2]
```

Figure 8: DistilBERT Experiments Phase 2 Configurations

The executions of all combinations took in total 53 minutes and 1 hr 36 minutes for Phase 1 and 2, respectively.

4.2 RoBERTa

For Experiments Phase 1 the configurations, fixed and changeable can be found in Figure 9:

```
# Fixed LoRA parameters
rank = 8
target_matrices = ["attention.self.query", "attention.self.key", "attention.self.value"]
# target_matrices = ["attention.self.query", "attention.self.key", "attention.self.value", "attention.output.dense"]
lora_alpha = 16
lora_dropout = 0.1

# Changing hyperparams for batch size, epochs and learning rates
batch_sizes = [8, 16]
epochs_list = [3, 5]
learning_rates = [3e-5, 1e-4]

training_dropout = 0.1 # Fixed
```

Figure 9: RoBERTa Experiments Phase 1 Configurations

For the Experiments Phase 2, the configurations can be found in Figure 10:

```
# Fixed parameters for batch size and epochs, etc
fixed_batch_size = 8
fixed_epochs = 5
fixed_learning_rate = 1e-4
training_dropout = 0.1

# LoRA parameter combinations
ranks = [8, 16]
target_matrices_list = [
    ["attention.self.query"],
    ["attention.self.query", "attention.self.key"],
    ["attention.self.query", "attention.self.key", "attention.self.value"]
]
lora_alpha = 16
lora_dropouts = [0.1, 0.2]
```

Figure 10: RoBERTa Experiments Phase 2 Configurations

The executions of all combinations took in total 24 minutes and 43 minutes for Phase 1 and 2, respectively.

4.3 ALBERT

For Experiments Phase 1 the configurations, fixed and changeable can be found in Figure 11:

```
# Fixed LoRA parameters
rank = 8
target_matrices = ["attention.query", "attention.key", "attention.value"]
lora_alpha = 16
lora_dropout = 0.1

# Changing hyperparams for batch size, epochs and learning rates
batch_sizes = [4, 8]
epochs_list = [3, 5]
learning_rates = [3e-5, 1e-4]

training_dropout = 0.1 # Fixed
```

Figure 11: ALBERT Experiments Phase 1 Configurations

For the Experiments Phase 2, the configurations can be found in Figure 12:

```
# Fixed parameters for batch size and epochs, etc
fixed_batch_size = 16
fixed_epochs = 5
fixed_learning_rate = 1e-4
training_dropout = 0.1

# LoRA parameter combinations
ranks = [8, 16]
target_matrices_list = [["attention.query"], ["attention.query", "attention.key"], ["attention.query", "attention.key", "attention.value"]]
lora_alpha = 16
lora_dropouts = [0.1, 0.2]
```

Figure 12: ALBERT Experiments Phase 2 Configurations

The executions of all combinations took in total 2 hours and 3 hours for Phase 1 and 2, respectively.

4.4 GPT2

For Experiments Phase 1 the configurations, fixed and changeable can be found in Figure 13:

```
# Fixed LoRA parameters
rank = 8
target_matrices = ["attn.c_attn", "attn.c_proj"]
lora_alpha = 16
lora_dropout = 0.1

# Changing hyperparams for batch size, epochs and learning rates
batch_sizes = [16, 32]
epochs_list = [3, 5]
learning_rates = [3e-5, 1e-4]

training_dropout = 0.1 # Fixed
```

Figure 13: GPT2 Experiments Phase 1 Configurations

For the Experiments Phase 2, the configurations can be found in Figure 14:

```
# Fixed parameters for batch size and epochs, etc
fixed_batch_size = 16
fixed_epochs = 3
fixed_learning_rate = 1e-4
training_dropout = 0.1

# LoRA parameter combinations
ranks = [8, 16]
target_matrices_list = [["attn.c_attn"], ["attn.c_proj"], ["attn.c_attn", "attn.c_proj"]]
lora_alpha = 16
lora_dropouts = [0.1, 0.2]
```

Figure 14: GPT2 Experiments Phase 2 Configurations

The executions of all combinations took over 4 hours for Phase 1 but by just one-fourth of the combinations for Phase 2 it had been run for almost 1 hr 30 minutes.

4.5 Experimentation Results

So keeping the LoRA configurations fixed with rank as 8, alpha as 16, LoRA dropout as 0.1 and target matrices all Query, Key and Value (Attn and Proj for GPT2), the testing results can be found in Table 2.

Model	Batch Size	Epochs	Learning Rate	Accuracy	Precision	Recall	F1-Score
DistilBERT	16	3	3.00E-05	0.87	0.87	0.87	0.87
DistilBERT	16	3	0.0001	0.89	0.89	0.89	0.89
DistilBERT	16	5	3.00E-05	0.89	0.89	0.89	0.89
DistilBERT	16	5	0.0001	0.90	0.90	0.90	0.90
DistilBERT	32	3	3.00E-05	0.84	0.84	0.84	0.84
DistilBERT	32	3	0.0001	0.88	0.88	0.88	0.88
DistilBERT	32	5	3.00E-05	0.87	0.87	0.87	0.87
DistilBERT	32	5	0.0001	0.89	0.89	0.89	0.89
RoBERTa	8	3	3.00E-05	0.87	0.87	0.87	0.87
RoBERTa	8	3	0.0001	0.89	0.89	0.89	0.89
RoBERTa	8	5	3.00E-05	0.88	0.88	0.88	0.88
RoBERTa	8	5	0.0001	0.89	0.89	0.89	0.89
RoBERTa	16	3	3.00E-05	0.88	0.88	0.88	0.88
RoBERTa	16	3	0.0001	0.89	0.89	0.89	0.89
RoBERTa	16	5	3.00E-05	0.88	0.88	0.88	0.88
RoBERTa	16	5	0.0001	0.89	0.89	0.89	0.89
ALBERT	4	3	3.00E-05	0.82	0.82	0.82	0.82
ALBERT	4	3	0.0001	0.91	0.91	0.91	0.91
ALBERT	4	5	3.00E-05	0.90	0.90	0.90	0.90
ALBERT	4	5	0.0001	0.92	0.92	0.92	0.92
ALBERT	8	3	3.00E-05	0.66	0.66	0.66	0.65
ALBERT	8	3	0.0001	0.90	0.90	0.90	0.90
ALBERT	8	5	3.00E-05	0.85	0.85	0.85	0.85
ALBERT	8	5	0.0001	0.92	0.92	0.92	0.92
GPT2	16	3	3.00E-05	0.73	0.74	0.73	0.73
GPT2	16	3	0.0001	0.90	0.90	0.90	0.90
GPT2	16	5	3.00E-05	0.84	0.84	0.84	0.84
GPT2	16	5	0.0001	0.90	0.90	0.90	0.90
GPT2	32	3	3.00E-05	0.74	0.74	0.74	0.74
GPT2	32	3	0.0001	0.90	0.90	0.90	0.90
GPT2	32	5	3.00E-05	0.84	0.84	0.84	0.84
GPT2	32	5	0.0001	0.90	0.90	0.90	0.90

Table 2: Test Results of Experiments Phase 1 across PLMs

While then keeping training hyperparameters fixed with batch size 16 (8 for RoBERTa), epochs as 5 and learning rate at 0.0001 and changing LoRA configurations (kept alpha as 16 throughout) the results on Test can be found in Table 3. Please note that due to long training and execution time, we could not fully continue the Phase 2 experimentation for GPT2; how ever much of it was done, is shown in the Table 3.

Model	Rank	LoRA Dropout	Target Matrices	Accuracy	Precision	Recall	F1-Score
DistilBERT	8	0.1	Q	0.89	0.89	0.89	0.89
DistilBERT	8	0.2	Q	0.89	0.89	0.89	0.89
DistilBERT	8	0.1	Q, K	0.89	0.89	0.89	0.89
DistilBERT	8	0.2	Q, K	0.89	0.89	0.89	0.89
DistilBERT	8	0.1	Q, K, V	0.90	0.90	0.90	0.90
DistilBERT	8	0.2	Q, K, V	0.90	0.90	0.90	0.90
DistilBERT	16	0.1	Q	0.90	0.90	0.90	0.90
DistilBERT	16	0.2	Q	0.90	0.90	0.90	0.90
DistilBERT	16	0.1	Q, K	0.90	0.90	0.90	0.90
DistilBERT	16	0.2	Q, K	0.90	0.90	0.90	0.90
DistilBERT	16	0.1	Q, K, V	0.90	0.90	0.90	0.90
DistilBERT	16	0.2	Q, K, V	0.89	0.89	0.89	0.89
RoBERTa	8	0.1	Q	0.89	0.89	0.89	0.89
RoBERTa	8	0.2	Q	0.89	0.89	0.89	0.89
RoBERTa	8	0.1	Q, K	0.89	0.89	0.89	0.89
RoBERTa	8	0.2	Q, K	0.89	0.89	0.89	0.89
RoBERTa	8	0.1	Q, K, V	0.89	0.89	0.89	0.89
RoBERTa	8	0.2	Q, K, V	0.89	0.89	0.89	0.89
RoBERTa	16	0.1	Q	0.90	0.90	0.90	0.90
RoBERTa	16	0.2	Q	0.90	0.90	0.90	0.90
RoBERTa	16	0.1	Q, K	0.90	0.90	0.90	0.90
RoBERTa	16	0.2	Q, K	0.90	0.90	0.90	0.90
RoBERTa	16	0.1	Q, K, V	0.90	0.90	0.90	0.90
RoBERTa	16	0.2	Q, K, V	0.89	0.89	0.89	0.89
ALBERT	8	0.1	Q	0.82	0.82	0.82	0.81
ALBERT	8	0.2	Q	0.82	0.82	0.82	0.82
ALBERT	8	0.1	Q, K	0.87	0.87	0.87	0.87
ALBERT	8	0.2	Q, K	0.87	0.87	0.87	0.87
ALBERT	8	0.1	Q, K, V	0.90	0.90	0.90	0.90
ALBERT	8	0.2	Q, K, V	0.90	0.90	0.90	0.90
ALBERT	16	0.1	Q	0.91	0.91	0.91	0.91
ALBERT	16	0.2	Q	0.91	0.91	0.91	0.91
ALBERT	16	0.1	Q, K	0.91	0.91	0.91	0.91
ALBERT	16	0.2	Q, K	0.91	0.91	0.91	0.91
ALBERT	16	0.1	Q, K, V	0.90	0.90	0.90	0.90
ALBERT	16	0.2	Q, K, V	0.90	0.90	0.90	0.90
GPT2	8	0.1	Attn	0.88	0.88	0.88	0.88
GPT2	8	0.2	Attn	0.88	0.88	0.88	0.88
GPT2	8	0.1	Proj	0.90	0.90	0.90	0.90
GPT2	8	0.2	Proj	0.91	0.91	0.91	0.91
GPT2	8	0.1	Attn, Proj	0.89	0.89	0.89	0.89

Table 3: Test Results of Experiments Phase 2 across PLMs

5 Final finetuned PLMs

5.1 Final Configurations

After eyeballing, observing training and validation losses, and evaluation metrics, for each PLM, and keeping in mind memory constraints, the best configurations were decided and can be found in Table 4. It must be noted that training dropout remained fixed throughout at 0.1 and LoRA alpha fixed at 16.

	DistilBERT	**RoBERTa**	**ALBERT**	**GPT2**
Batch size	16	16	16	16
Learning rate	0.0001	0.0001	0.0001	0.0001
Epochs	5	5	5	5
Rank	16	16	16	8
Target Matrices	Q, K	Q	Q, K	Attn, Proj
LoRA Dropout	0.2	0.2	0.2	0.1

Table 4: Final Configurations for Finetuning

5.2 Final Finetuning Results

The stated configurations yield the results found in Table 5 while their parameter training and memory related information is contained in Table 6. To reiterate, except for GPT2 which was final finetuned for 5% of the training and test datas, all models have been final finetuned on the complete datas.

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
DistilBERT	0.92	0.92	0.92	0.92
RoBERTa	0.95	0.95	0.95	0.95
ALBERT	0.93	0.93	0.93	0.93
GPT2	0.91	0.91	0.91	0.91

Table 5: Test Results of Finetuned PLMs

Model	**Total Params**	**Trainable Params**	**% Trainable**	**GPU Memory**	**Training Time**
DistilBERT	67,842,052	887,042	1.31%	259.88 MB	78 min
RoBERTa	125,534,212	887,042	0.71%	480.12 MB	83 min
ALBERT	11,735,812	50,690	0.43%	44.78 MB	167 min
GPT2	124,885,248	443,904	0.36%	489.17 MB	82 min

Table 6: PEFT Information of the Finetuning

5.3 Finetuned PLMs Pushed to HuggingFace

As a culmination of the finetuning, the PLMs were also pushed to HuggingFace as evidenced in Figure 15.

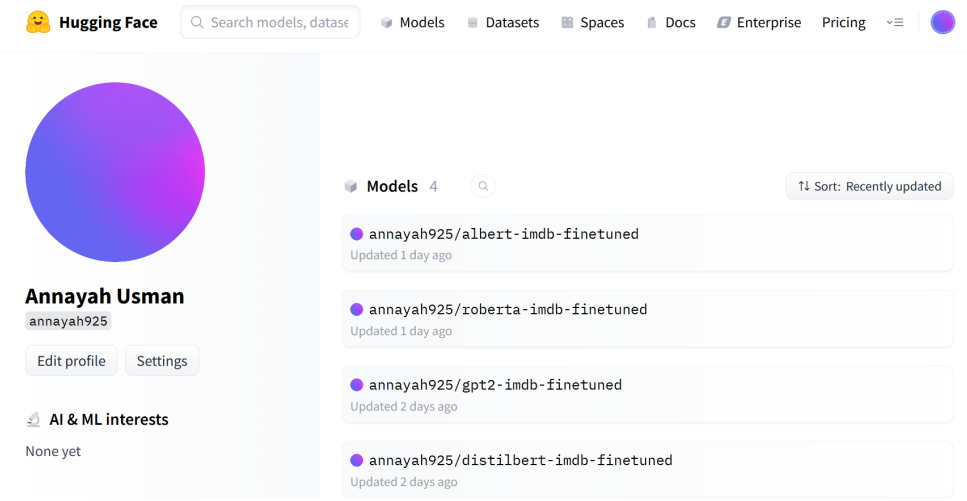

Figure 15: Finetuned PLMs on HuggingFace

6 Results and Discussion

The baseline's accuracy stood at 0.89 while the best performing ML model, Logistic Regression stood at par with the same accuracy. With different TFIDF vectorization configurations, the trend was observed that the more the ngram range is widened and/or the less the cap on the maximum features, the better the model will correctly classify. The incorporation of embedding did increase accuracies or F1 score, but not for some already well-peforming models such as Logistic whose performance stayed on 0.89. However, the poorer performing models such a k-NN saw a steep increase from ranging between accuracy of 0.70 and 0.73 to reaching 0.78 by the inclusion of the embedding.

From the finetuning experimentation results, one observation is consistent across all PLMs: the a relatively larger larger learning rate improves classification. That is why between the options of 0.00003 and 0.0001, the latter consistently contributed to giving better results. Increasing the batch size and its effect on performance still had a lot to do with the larger learning rate, however, in general batch sizes of 8 and 16 yielded surpassing benchmarks scores –crossing 0.90 accuracy and F1 scores and reaching at max 0.91. Some interesting cases included ALBERT giving performance of 0.91 with batch size of 4, learning rate 0.0001 and with 3 epochs. While in one instance of the initial run of the RoBERTa experimentation (which had to be redone due to going out

of memory) when batch sizes were initially chosen as 16 and 32, the accuracy and F1 score had crossed 0.93 consistently during 5 epochs. Hence, it was also noted the more the epochs, the better because the model has more room for training and learning to be able to predict new data. Therefore, 5 epochs was the viable option across all models.

By changing configurations of LoRA and observing their results, it was found that of the two options for rank (8 and 16), the higher value pushed accuracy consistently towards and above 0.90, over 0.91 for ALBERT in particular, but under 0.92. While a higher LoRA dropout rate, contributed to upgrading performance. For example, for the Phase 2 experimentations run for GPT2, it was seen that a with the lower rank, 8, a higher dropout rate of 0.2 touched accuracy and F1 score to 0.91.

The most interesting aspect of observing and deciding LoRA configurations was the target modules/matrices. Though for this project, the options were limited to only the Q, K and V matrices and their sequential combinations. There were several cases across all models that noted that one doesn't need to target and add all the layers or those matrices for finetuning. Even tuning one of the three may result in similar accuracy to finetuning all three. For DistilBERT, RoBERTa and ALBERT it was certainly the case that only finetuning for the Query matrix yielded an accuracy of 0.91. While for GPT2 just the Projection matrix yielded similar results. Therefore, in our final finetuning, the Value matrix was not targeted and the finetuned models yielded exemplary scores, all surpassing the ML models and the baseline accuracies.

In terms of Parameter Efficient Fine Tuning (PEFT), it can be seen empirically that by LoRA and the process of PEFT, overall we are finetuning the PLMs by training ¡1% of the total weights. It is a surprising observation, that though RoBERTa is larger model than DistilBERT and of course, ALBERT, the latter took twice as much time for training than RoBERTa. This could be due to ALBERT's architechture having shared parameters, and perhaps some memory efficient configurations set for each of them in the program/code.

Nonetheless, the results suggests the reign of the BERT subfamily amongst the PLMs, with RoBERTa at the top. However, GPT2 was just final finetuned on 5% of the training and test data and yielded a 0.91 accuracy. Therefore, with larger memory and computational resources, the finetuning of these PLMs could surpass expectations and the current benchmarks now set as a result of this project.

7 Future Work

The following points can be considered for further work from this point onwards:

- Using your own finetuned models, that are now pushed to HuggingFace, to stack them in an ensemble fashion and then perform classification on test
- Methodology for hyperparameter tuning for improved guidance towards the best configurations
- Employing adaptive rank selection for determining the rank that optimizes the tradeoff between model performance and computational efficiency.
- Running model for more epochs and observing the changes in results.