Fine-Tuning Models using Python and Hugging Face's software ecosystem for sentiment analysis.

Shilpa Bombale 202291958

Introduction

Fine-tuning an existing model demands more computational resources and technical expertise compared to using a pre-trained model out-of-the-box. However, smaller fine-tuned models can surpass larger pre-trained base models in specific use cases, even with advanced prompt engineering. Moreover, with the abundance of open-source LLM resources available today, fine-tuning a model for custom applications has become more accessible than ever.

This project involves using the Hugging Face ecosystem to fine-tune **distilbert-base-uncased,** a ~70M parameter model and **albert-base-v2**, a ~11.8M parameter model, both based on **BERT** for text classification, identifying 'positive' and 'negative' sentiments.

We will load our training and validation data from the Hugging Face datasets library. This dataset comprises 2000 movie reviews, with 1000 reviews allocated for training and 1000 for validation. Each review has a binary label indicating whether it is positive or not.

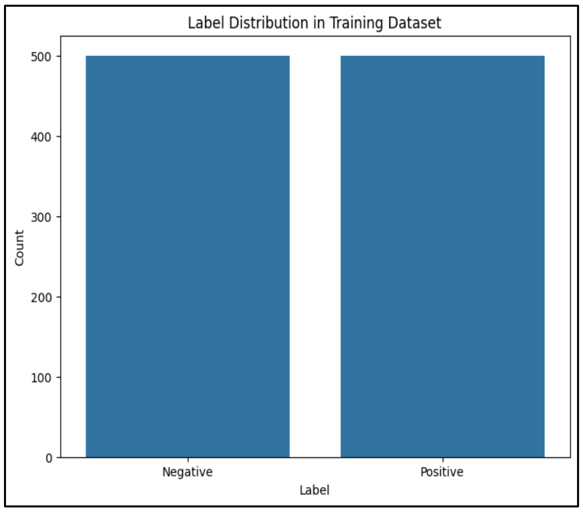


Fig 1: Equal distribution of sentiments.

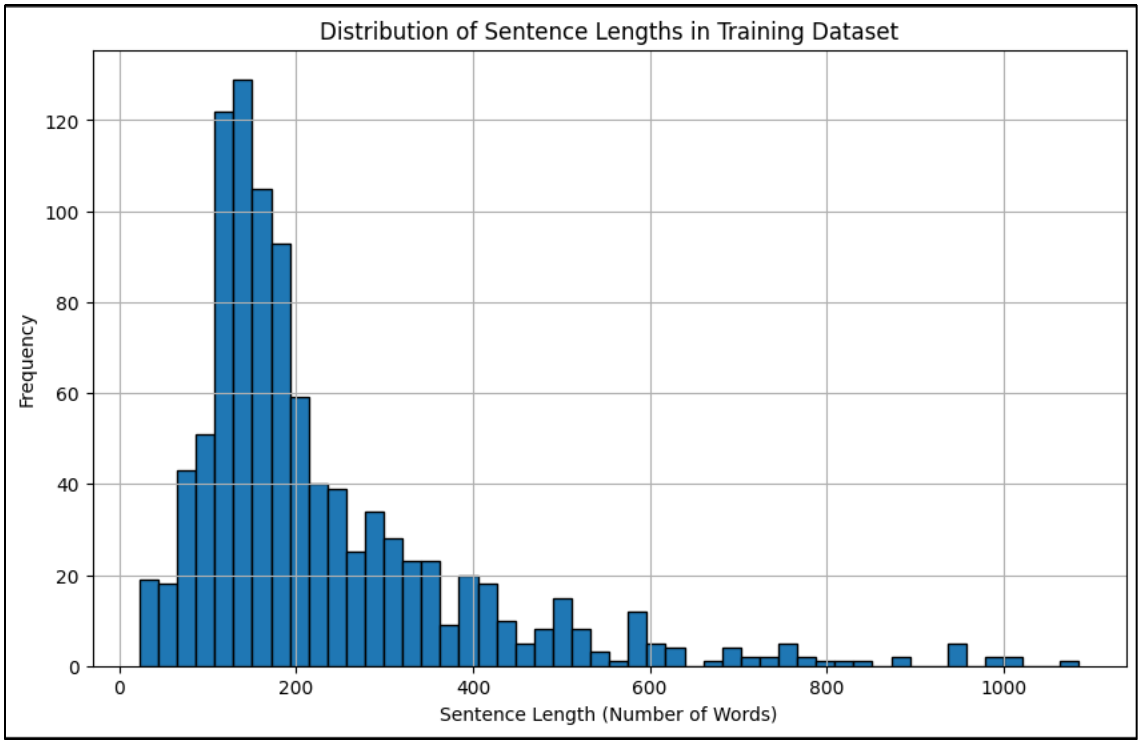


Fig 2: Dataset visualization

Methodology

Fine-tuning is a common technique in machine learning. It involves taking a pre-trained model and further training it for a specific task. This method makes use of the broad knowledge the model gained from its initial training, typically on a large and varied dataset, and tailors it to meet the needs of a particular application.

While fine-tuning is powerful, it has a significant drawback: it's resource intensive. Large models demand substantial computational power, not only for their initial training but also for adapting them to new tasks. This is where **parameter-efficient fine-tuning** becomes crucial; otherwise, fine-tuning large language models on our local machines would be impractical.

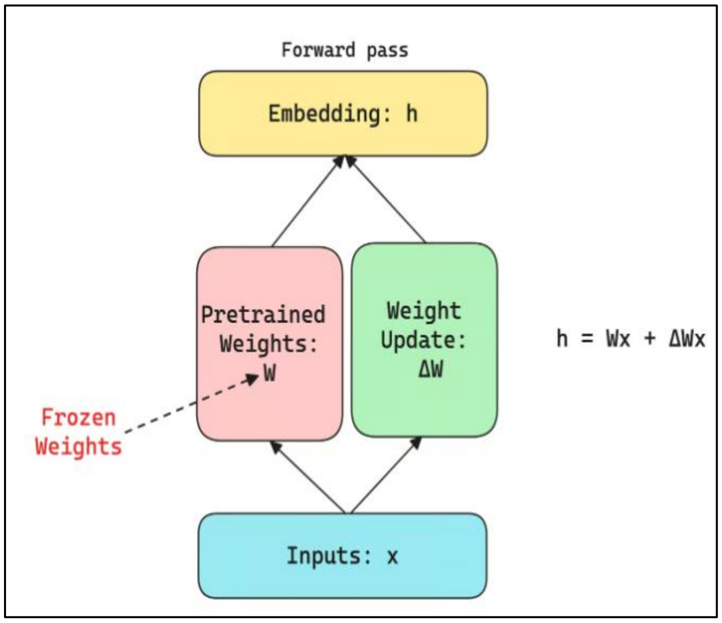


Fig 3: Representation of the finetuning process

To set the stage for **LoRA**, consider this: if we only update ΔW without modifying the original pre-trained weights W (which remain frozen or have req_grads = False), we can save a lot of GPU memory. Moreover, if these updated weights are just a small fraction of the original W, the savings are even greater.

Pre-trained models have a very low intrinsic dimension, or low rank, which means they can be represented with matrices of much lower dimensions. The rank of a matrix represents the number of linearly independent rows or columns it has.

During adaptation or fine-tuning, ΔW (of dimension A×B) also has a low rank and can be decomposed into two matrices, WA (of dimension A×r) and WB (of dimension r×B), where r is the rank of matrix ΔW.

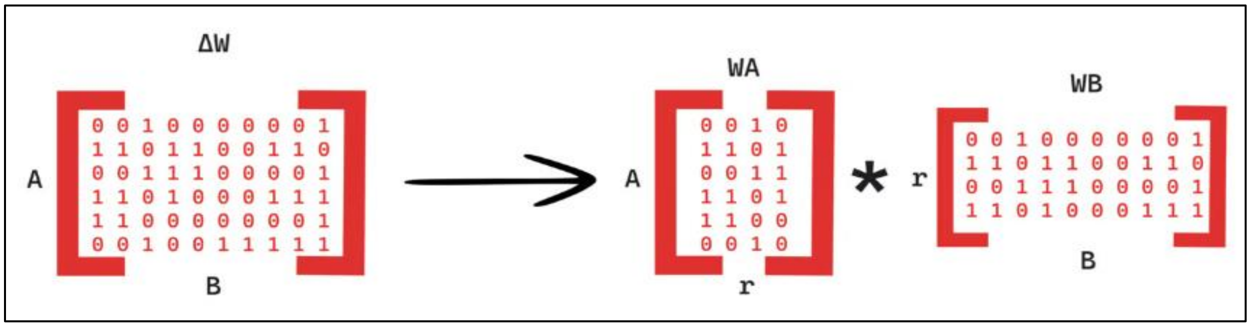


Fig 4: Low Rank Decomposition

For example, if ΔW has a dimension of 100×100 and a rank of just 5, it can be decomposed into two matrices with dimensions 100×5 and 5×100. This effectively reduces the number of trainable parameters from 10,000 to just 1,000.

In this setup, r becomes a hyper-parameter that needs to be tuned during training. The key point is that WA and WB can effectively represent ΔW while having a significantly lower rank and therefore fewer parameters to fine-tune.

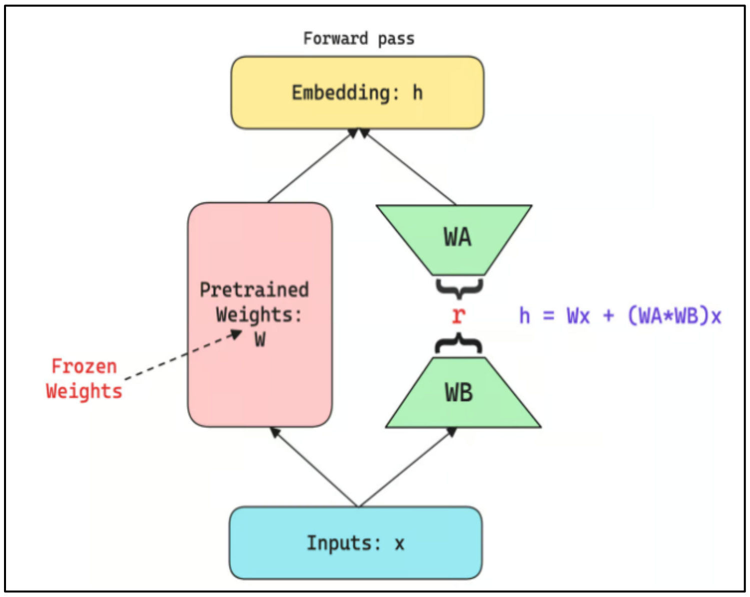


Fig 5: Fine-Tuning using LoRA

Training Process

This section outlines the code used for training a sequence classification model with fine-tuning and Low-Rank Adaptation (LoRA). The code leverages various libraries and frameworks, including Hugging Face's transformers, peft, and evaluate.

1. **Loading the Dataset**
   The code begins by loading a dataset from the Hugging Face dataset repository. In this case, it uses the IMDb dataset (truncated version) for sentiment analysis.

2. **Model and Tokenizer Setup**
   The code initializes a pre-trained model (distilbert-base-uncased) for sequence classification and sets up a tokenizer for text processing. Label mappings are defined to convert between numerical labels and human-readable text.

3. **Data Collation and Metrics**
   A data collator is created to handle dynamic padding of input sequences. The accuracy metric is loaded from the evaluate library. An evaluation function is defined to compute the accuracy during model training.

4. **Model Initialization with LoRA**
   LoRA (Low-Rank Adaptation) configuration is defined, including rank and other parameters. The model is then adapted using LoRA to reduce the number of trainable parameters.

5. **Training Setup**
   Training arguments are defined, including learning rate, batch size, and number of epochs. A Trainer object is created to handle the training process, which includes training and evaluation datasets, data collation, and metric computation.

6. **Training and Evaluation**

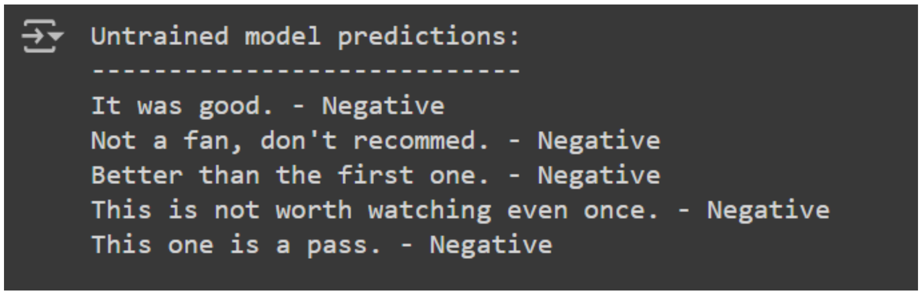
   The model is trained using the Trainer object. After training, the model is moved to the CPU (or another specified device) for inference. Example texts are then evaluated using the trained model to demonstrate its performance.

Outputs

In this section, we present and analyze the performance of the model both before and after training. The evaluation focuses on the model's predictions on a set of example texts and various performance metrics.

1. **Performance of untrained model**

   Before training the model, we run predictions on a sample set of texts to understand its initial performance. This helps establish a baseline for comparison after the fine-tuning process.



```
Untrained model predictions:
----------------------------
It was good. - Negative
Not a fan, don't recommed. - Negative
Better than the first one. - Negative
This is not worth watching even once. - Negative
This one is a pass. - Negative
```

Fig 5: Performance of untrained models

   The untrained model tends to predict "Negative" for all the examples, indicating it has not yet learned to distinguish between positive and negative sentiments. This underscores the need for fine-tuning to improve the model's predictive capabilities.

2. **Training the Model**

   The model is trained on the IMDb dataset using the specified training parameters and configuration. This process involves multiple epochs, where the model is trained iteratively, and evaluations are conducted at the end of each epoch to identify its performance metrics.

Accuracy: The proportion of correctly identified instances out of all instances.

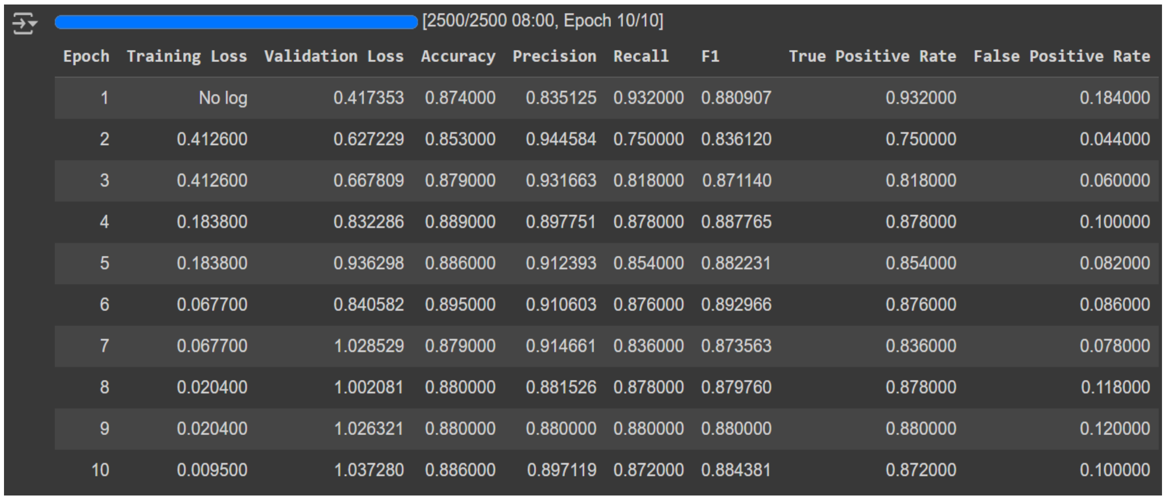
Precision: The proportion of true positives out of all predicted positives.

Recall (True Positive Rate): The proportion of true positives out of all actual positives.

F1 Score: The harmonic mean of Precision and Recall.

True Positive Rate (TPR): Indicates how well the model identifies positive instances.

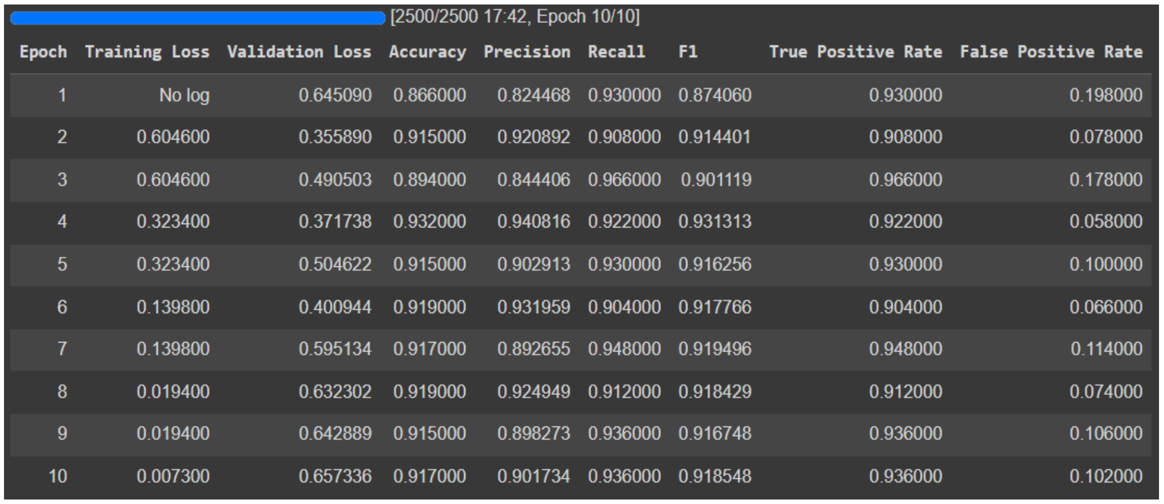
False Positive Rate (FPR): Indicates how often the model incorrectly identifies negative instances as positive.



[2500/2500 08:00, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1	True Positive Rate	False Positive Rate
1	No log	0.417353	0.874000	0.835125	0.932000	0.880907	0.932000	0.184000
2	0.412600	0.627229	0.853000	0.944584	0.750000	0.836120	0.750000	0.044000
3	0.412600	0.667809	0.879000	0.931663	0.818000	0.871140	0.818000	0.060000
4	0.183800	0.832286	0.889000	0.897751	0.878000	0.887765	0.878000	0.100000
5	0.183800	0.936298	0.886000	0.912393	0.854000	0.882231	0.854000	0.082000
6	0.067700	0.840582	0.895000	0.910603	0.876000	0.892966	0.876000	0.086000
7	0.067700	1.028529	0.879000	0.914661	0.836000	0.873563	0.836000	0.078000
8	0.020400	1.002081	0.880000	0.881526	0.878000	0.879760	0.878000	0.118000
9	0.020400	1.026321	0.880000	0.880000	0.880000	0.880000	0.880000	0.120000
10	0.009500	1.037280	0.886000	0.897119	0.872000	0.884381	0.872000	0.100000

Fig 6: Evaluation Output for distilbert-base-uncased



[2500/2500 17:42, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1	True Positive Rate	False Positive Rate
1	No log	0.645090	0.866000	0.824468	0.930000	0.874060	0.930000	0.198000
2	0.604600	0.355890	0.915000	0.920892	0.908000	0.914401	0.908000	0.078000
3	0.604600	0.490503	0.894000	0.844406	0.966000	0.901119	0.966000	0.178000
4	0.323400	0.371738	0.932000	0.940816	0.922000	0.931313	0.922000	0.058000
5	0.323400	0.504622	0.915000	0.902913	0.930000	0.916256	0.930000	0.100000
6	0.139800	0.400944	0.919000	0.931959	0.904000	0.917766	0.904000	0.066000
7	0.139800	0.595134	0.917000	0.892655	0.948000	0.919496	0.948000	0.114000
8	0.019400	0.632302	0.919000	0.924949	0.912000	0.918429	0.912000	0.074000
9	0.019400	0.642889	0.915000	0.898273	0.936000	0.916748	0.936000	0.106000
10	0.007300	0.657336	0.917000	0.901734	0.936000	0.918548	0.936000	0.102000

Fig 7: Evaluation Output for albert-base-v2

These metrics show that the model performs significantly better after training, with balanced precision and recall.

ROC Curve:

Create an ROC curve by thresholding continuous predictions to treat the regression task as a binary classification problem. Calculate the AUC and plot the ROC curve.

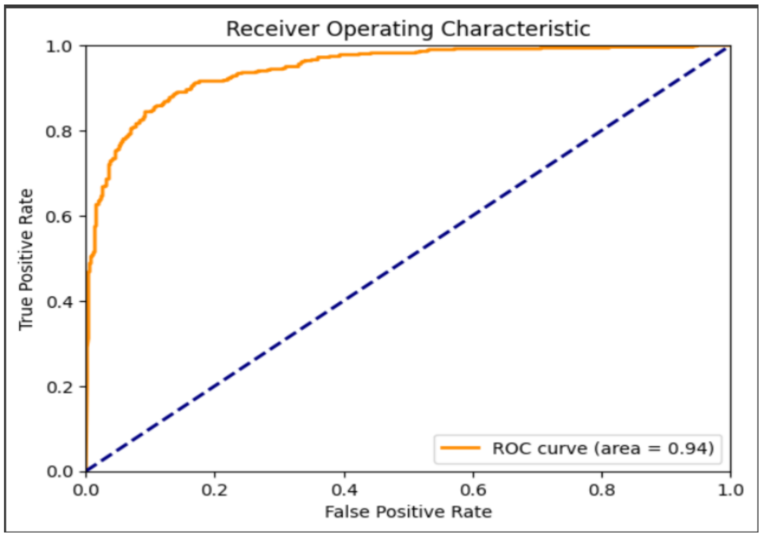


Fig: ROC curve for distilbert-base-uncased

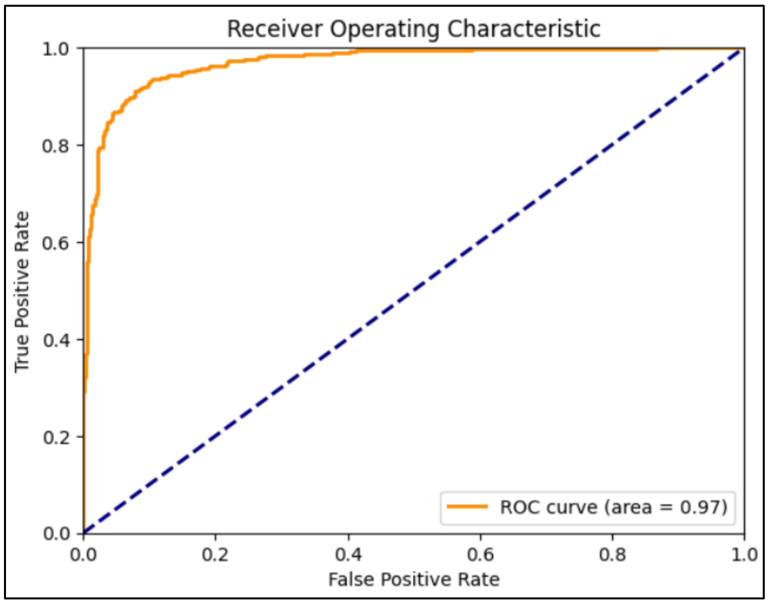
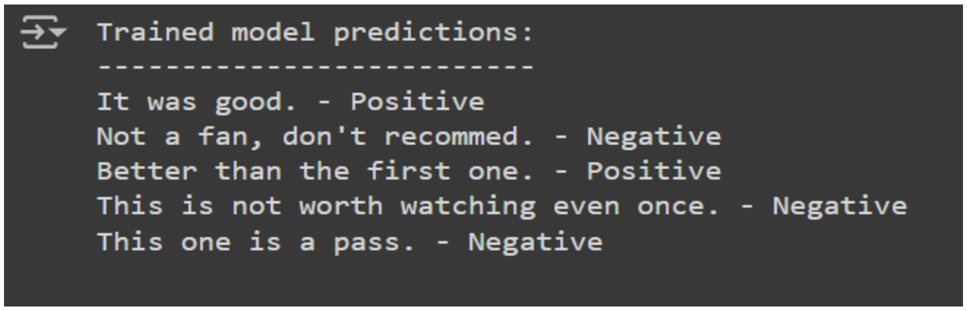


Fig: ROC curve for albert-base-v2

3. **Trained Model Predictions**

After training, we again run predictions on the same set of example texts to compare the results with the untrained models.



```
Trained model predictions:
--------------------------
It was good. - Positive
Not a fan, don't recommed. - Negative
Better than the first one. - Positive
This is not worth watching even once. - Negative
This one is a pass. - Negative
```

Figure 7: Performance of trained models

The trained models correctly identifies both positive and negative sentiments in the example texts. This demonstrates the effectiveness of the fine-tuning process, which enables the model to learn and distinguish between different sentiments more accurately.

Conclusion:

In this project, we successfully fine-tuned the distilbert-base-uncased and albert-base-v2 models using the Hugging Face ecosystem to perform sentiment analysis on movie reviews. Our methodology involved leveraging pre-trained models and further adapting them to our specific task using advanced techniques such as Low-Rank Adaptation (LoRA). The results demonstrate that fine-tuning, despite its resource-intensive nature, can significantly enhance the performance of smaller models in specific applications. Our fine-tuned models outperformed their initial versions, showing a marked improvement in accuracy, precision, recall, and F1 scores. Additionally, the ROC curves for both models indicated a strong ability to distinguish between positive and negative sentiments post-training.

This project highlights the effectiveness of fine-tuning and parameter-efficient techniques in optimizing model performance for custom tasks. It underscores the potential of leveraging open-source resources and advanced training methodologies to make sophisticated machine learning accessible and practical.

References

[1] Victor Sanh, Lysandre Debut, Julien Chaumond, Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. ArXiv, abs/1910.01108, 2020

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In NAACL-HLT, 2018.

[3] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar S. Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. ArXiv, abs/1907.11692, 2019

[4] Understanding LoRA: Low-rank Adaption of Large Language modeld by Akshay Pachaar. https://mlspring.beehiiv.com/p/understanding-lora-lowrank-adaption-large-language-models

[5] Fine-Tuning Large Language Models by Shaw Talebi https://towardsdatascience.com/fine-tuning-large-language-models-llms-23473d763b91