CISC 691: Assignment 03 By Vikramvelupandian Palani

Fine-Tuning Documentation for Sentiment Analysis:

The approach, objectives, and details of the implementation for fine-tuning a reduced *DistilBert* model on the IMDB sentiment analysis task. It also covers model performance evaluation, challenges encountered, and the solutions applied.

Motivation for Fine-Tuning in Sentiment Analysis:

Leveraging Pre-trained Knowledge: Through fine-tuning you can utilize models that have been pre-trained on extensive general text datasets. Models such as BERT and GPT-3 along with Roberta represent popular choices. The knowledge obtained from pre-training is tailored for sentiment analysis through selective updating of model weights.

Confidence in efficiency and Resource Savings: Fine-tuning proves to be an efficient solution since it delivers strong performance on specialized tasks while eliminating the need to utilize extensive labeled data and computational power.

Adaptability: The approach facilitates quick development and implementation across numerous practical applications including product service enhancement through customer review analysis and brand reputation management via social media sentiment understanding and user experience improvement through feedback survey analysis.

Training models from scratch requires extensive resources and data while fine-tuning pre-trained models achieves better efficiency.

Pre-Trained Representations: The model possesses existing capabilities in natural language comprehension which necessitates only small modifications for effective sentiment classification specialization.

Reduced Training Time: By utilizing fine-tuning the training time reduces because the model has already learned from extensive data.

Lower Computational Demand: Using fine-tuning reduces the necessity of vast datasets and high computational resources which creates accessibility for organizations with limited resources.

Practical Applications of Sentiment Analysis

Customer Feedback: Evaluate customer satisfaction levels through analysis of product reviews and service interactions.

Social Media Monitoring: Sentiment analysis enables businesses to comprehend how the public feels about brands, products or political happenings.

Market Research: Tracking consumer behavior trends alongside sentiment analysis helps shape effective marketing strategies.

Define Objectives:

Fine-Tuning Goals

Adapt a Pre-Trained Model: We aim to fine-tune a compressed *distilbert-base-uncased* model for positive and negative sentiment classification using a balanced IMDB dataset subset. The clear objective will direct our work and maintain our progress throughout the project.

Improve Task-Specific Performance: Fine-tune the model parameters to achieve precise sentiment prediction for new text inputs.

Target Performance Metrics

Accuracy: The percentage of correct predictions overall predictions.

Weighted F1 Score: The weighted F1 Score measures the harmonic meaning between precision and recall while considering the imbalance between classes. The dataset shows a major imbalance between classes which helps in providing an accurate performance evaluation. The model's effectiveness is assessed through metrics that are calculated throughout both training and evaluation phases.

Baseline Model and Target Improvements

Baseline Model: We utilize the original pre-trained DistilBert model as our baseline. The implementation uses a reduced model version with 3 layers instead of the original 6 to enhance training efficiency through a smaller hidden size. The baseline model provides the essential foundation needed for all subsequent improvement efforts.

Target Improvements: The fine-tuning process seeks to match the full-size model's performance metrics while creating a smaller model with faster execution for real-time applications.

Plan Implementation Steps:

Pre-Trained Model Selection

Model Choice: For model implementation we will use *distilbert-base-uncased* from Hugging Face. The code achieves a smaller footprint by using fewer layers and decreasing hidden size to accelerate both fine-tuning and inference processes.

Dataset and Preprocessing Approach

Dataset Selection: Researchers use the IMDB dataset which serves as a popular benchmark for sentiment analysis tasks.

Data Balancing and Filtering: The dataset undergoes filtering through the code to maintain equal numbers of positive and negative examples which helps prevent training bias.

Tokenization: Text data goes through tokenization by the *AutoTokenizer* which is tied to the base model. The process of tokenization uses padding and truncation methods to ensure input size uniformity.

Evaluation Metrics and Testing Strategy

Metrics Computation: The implementation tracks performance by measuring both accuracy and the weighted F1 score through (*sklearn.metrics*).

Evaluation Strategy: The system evaluates model performance after each epoch and stores the model with the highest performance. The evaluation function operates independently to test sample sentences and verify model generalization across new inputs.

Code Documentation and Explanations:

Logging Configuration

- **Function:** configure_logging
  Sets up both console and file logging to capture detailed debugging and execution flow information.

Model Loading and Reduction

- **Function:** load_base_model
  - ✓ Loads the pre-trained model and tokenizer.
  - ✓ Creates a reduced model configuration using DistilBertConfig (with fewer layers and a smaller hidden size).
  - ✓ Saves the reduced model locally for fine-tuning later.

Dataset Loading, Filtering, and Tokenization

- **Function:** load_sentiment_dataset
    - ✓ Downloads the IMDB dataset from Hugging Face.
    - ✓ Filters and balances the dataset to have an equal number of positive and negative samples.
    - ✓ Tokenize the text data using the associated tokenizer and saves the processed dataset to disk.

Training Loop and Evaluation

- **Function:** train_model
    - ✓ Prepares training and testing data using a custom SentimentDataset class.
    - ✓ Configures training arguments (including batch sizes, learning rate, number of epochs, and device settings).
    - ✓ Implements the training loop using the Hugging Face Trainer API, which also computes metrics at each epoch.
    - ✓ Saves the fine-tuned model and tokenizer after training completes.

Prediction and Evaluation of Sample Sentences

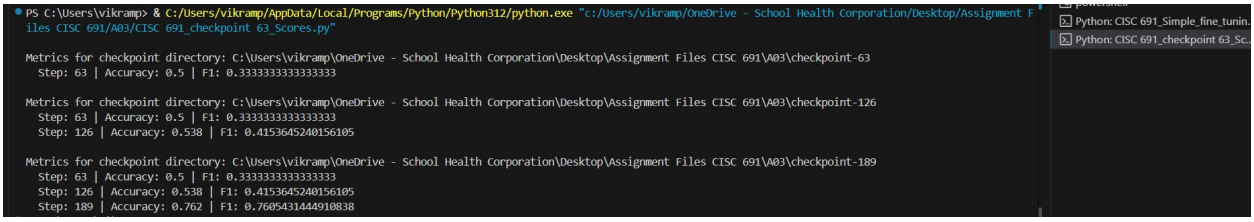
- **Functions:** predict_sentiment and evaluate_sample_sentences
    - ✓ predict_sentiment: Accepts a single text input, tokenizes it, and outputs a sentiment prediction.
    - ✓ evaluate_sample_sentences: Reads sample sentences from a JSON file and prints the predicted versus ground-truth sentiments for quick qualitative analysis.

Main Execution Flow

- **Function:** main()
  Orchestrates the entire workflow:
    1. Loads or creates the model and dataset.
    2. Triggers the training process.
    3. Runs predictions on test texts.
    4. Evaluates additional sample sentences from an external JSON file.

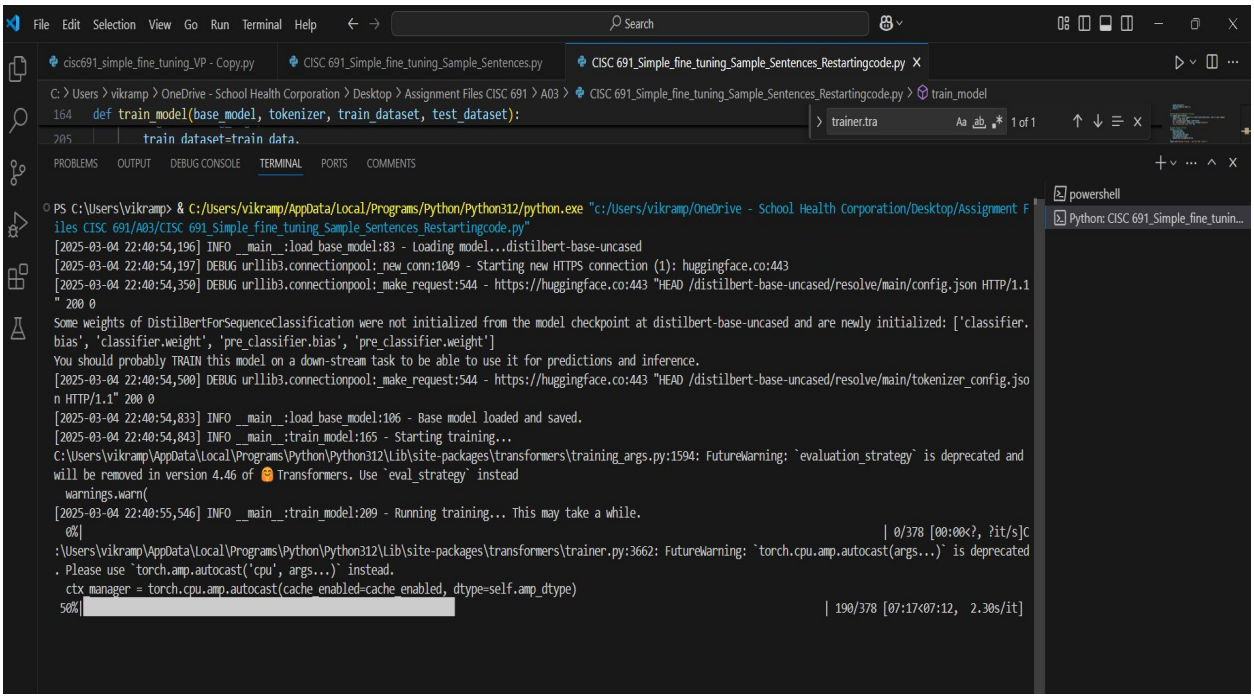
Analysis of Model Performance:

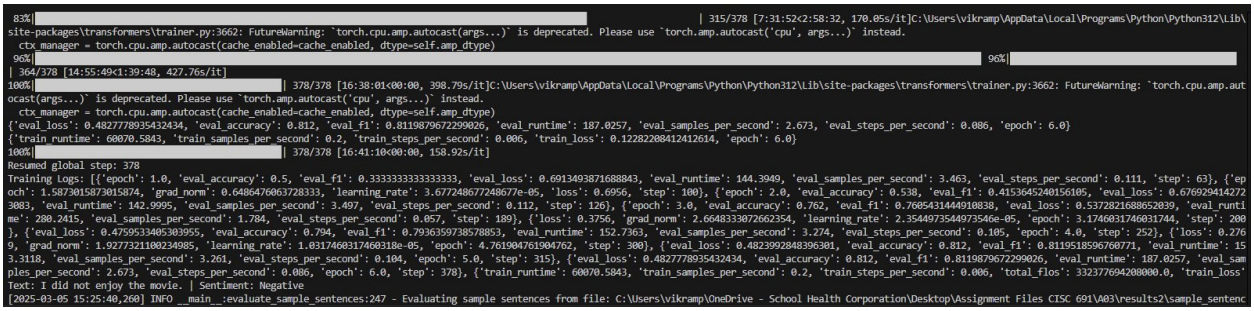
• ***Metrics Collection:*** Accuracy and F1 scores are recorded at each epoch's conclusion during the training process. These metrics provide insights into the classification capabilities of the model.



• ***Evaluation Results:*** The evaluation function outputs performance metrics after training which allows for a comparison between the fine-tuned model and baseline models.

• ***Logging Insights:*** Detailed logging enables users to monitor training progress and detect potential overfitting or underfitting problems. The below image shows that the code restarted from the last checkpoint.





Discussion of Challenges and Solutions:

Challenges

Data Imbalance and Noise: The initial dataset could demonstrate class imbalance or include inaccurate labels. To address class imbalance in the dataset we created a balanced subset that included both positive and negative reviews.

Model Size vs. Performance Trade-Off: The model performance could be affected when its size is reduced through a decrease in the number of layers and hidden dimensions. This compromise between model size and performance enabled quicker training and inference times which is essential for effective deployment.

Device Compatibility: The code performs a strong verification of Apple Silicon (MPS) compatibility while switching to CPU usage when necessary. The system remains adaptable to multiple hardware types because of the detailed attention given to device compatibility.

Tokenization and Sequence Length: The correct tokenization of text inputs with uniform padding and truncation was essential to prevent runtime errors while sustaining model performance.

Solutions

Balanced Sampling: The data loading function implements equal sampling for both positive and negative examples.

Logging and Monitoring: The project includes detailed logging as an essential element because it delivers valuable information about how the training process unfolds. This system feature helps identify performance and convergence problems which supports better model accuracy.

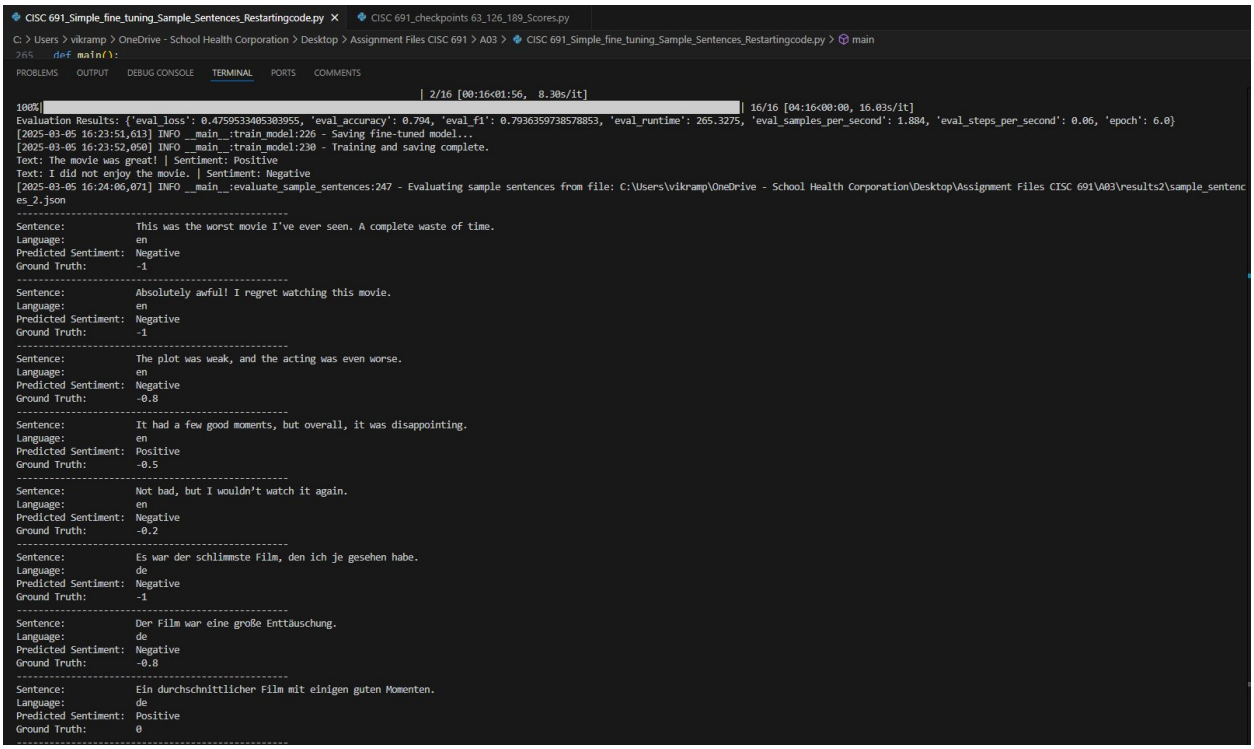
Flexible Device Assignment: Device checks represent a critical component of this project. The system directs the model to operate on optimal hardware resources which results in substantial efficiency improvements.

Modular Code Design: The project uses modular functions such as model loading, dataset processing, training, and prediction which simplifies the process of adjusting and troubleshooting different pipeline parts.

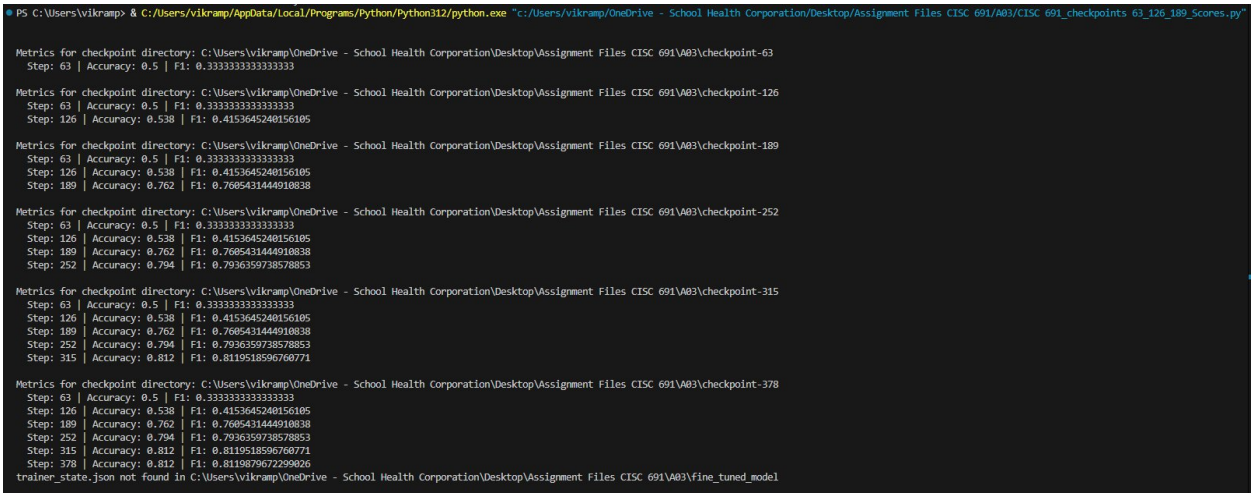
Conclusion and Results:

The process of fine-tuning a sentiment analysis system with a smaller DistilBert model. Through the integration of pre-trained models with balanced datasets and strict evaluation metrics the

system design ensures practical and efficient sentiment classification performance. Modular code design and comprehensive logging resolve the issues of data handling, model scaling, and device compatibility by delivering a solution that remains strong and versatile across various situations. As a result of successfully training the dataset and running the sample_sentences_2 file it provides clear results after running the file using the trained model. The final step of this entire project is to use this file to test our pre-trained model efficiently. The image below is the proof the model can predict the sentiment correctly.



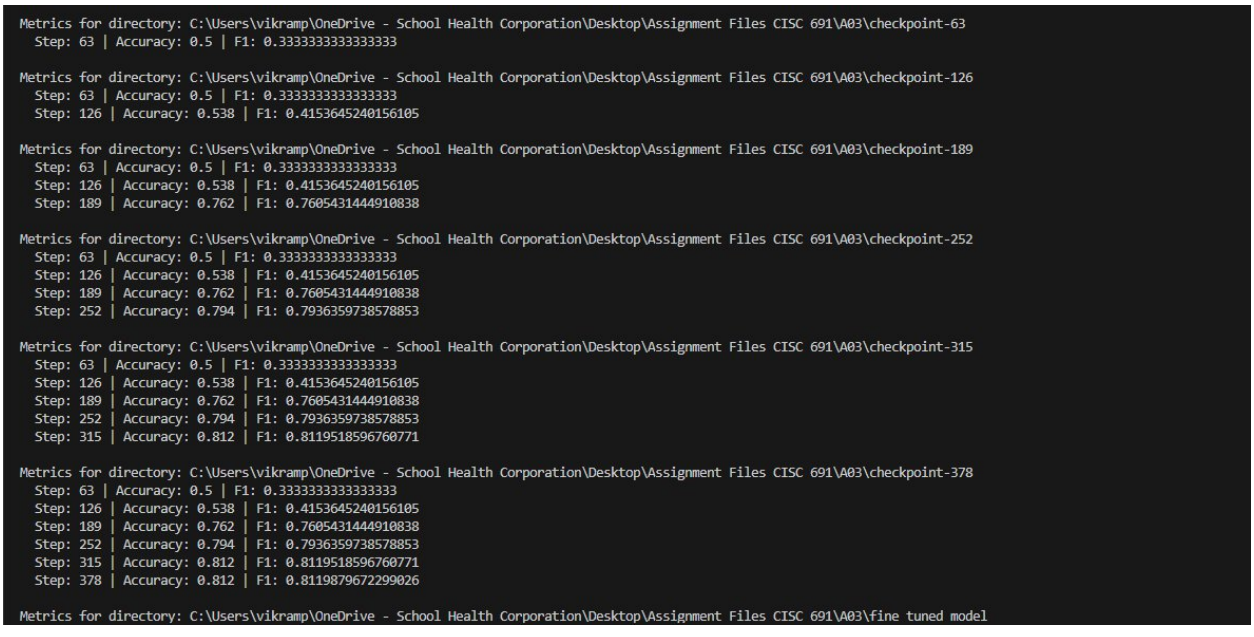
Below image gives you the results efficiency of the metrics for checkpoints



The checkpoints show a clear trend of improvement during training:

- **Checkpoint-63:**
  The initial evaluation shows baseline performance with 50% accuracy and an F1 score of 0.333.
- **Checkpoint-126:**
  A modest improvement is seen with accuracy rising to about 53.8% and the F1 score to approximately 0.415.
- **Checkpoint-189:**
  A significant jump occurs here with accuracy increasing to around 76.2% and F1 nearly matching at 0.7605.
- **Checkpoint-252:**
  Further improvement is observed, with metrics reaching about 79.4% accuracy and an F1 score of 0.7936.
- **Checkpoint-315:**
  Performance improves slightly more to roughly 81.2% accuracy and an F1 score of 0.812.
- **Checkpoint-378:**
  The metrics remain stable at approximately 81.2% for both accuracy and F1 score.

Below image gives you the results efficiency of the metrics of directory



Finally, obtained the final fine-tuned model results. Here's what happened:

1. **Resuming from Checkpoint 378:**
   The log shows “Resumed global step: 378,” meaning that the training run picked up from the last saved checkpoint (checkpoint-378). No additional training steps were taken beyond that point.

2. **Evaluation Metrics:**
   After resuming, the training script immediately proceeded to evaluation. The evaluation results show an accuracy of about 79.4% and an F1 score of roughly 0.7936. These metrics represent the performance of the model as of checkpoint 378, which is the final state of your fine-tuning.

3. **Saving and Inference:**
   The script then saved the fine-tuned model and ran predictions on sample sentences. The predictions (e.g., "The movie was great!" → Positive) confirm that the model from checkpoint 378 is now being used for inference.

In summary, by resuming from checkpoint 378 and running the evaluation right after, have effectively obtained and validated the final fine-tuned model results.