NLP ASSIGNMENT 2 - REPORT

Akshat Khare – 300342170

Yash Keswani – 300375526

Siddhant Tiwari - 300294512

Task Distribution

Akshat Khare	Yash Keswani	Siddhant Tiwari
1. Pre-process the dataset. 2. Train baseline classifier (Random Forest Classifier). 3. Evaluate the classifier and generate the prediction file.	1. Pre-process the dataset. 2. Fine-tune the transformer-based classifier (DistilBERT). 3. Evaluate the classifier and generate the prediction file.	1. Preprocess the dataset. 2. Fine-tune the generative LLM (BAAI-BGE-LLM) for classification. 3. Evaluate the classifier and generate the prediction file.

Part 1

Preprocessing of Dataset:

1. **Removal of stop words:** Stop words are common words in a language that do not carry significant meaning (conjunctions, articles and pronouns) and are often removed from text data during preprocessing.
2. **Tokenization:** Tokenization is the process of breaking down a text into smaller units, such as words or punctuation marks. In our case, we have used TF-IDF tokenizer which tokenizes the text data, calculates the TF and IDF values for each term, and constructs a TF-IDF matrix representing the importance of each term in each document relative to the entire document collection.
3. **Splitting the dataset:** Training dataset was split into 80% and 20% training and testing ratio respectively.

Training Baseline:

1. **Why Random Forest:** Random Forest Random Forest combines multiple decision trees to mitigate overfitting and improve generalization by aggregating their predictions. It effectively

captures complex non-linear relationships between input features and target variables, crucial for discerning intricate patterns in text classification tasks. The inherent averaging mechanism and limited depth per tree makes it less susceptible to overfitting and beneficial when handling noisy text data. Provides insights into the importance of individual features (words) for classification, aiding in identifying discriminative terms indicative of human or ChatGPT-generated text. Also efficiently manages high-dimensional feature spaces common in text data, making it suitable for large-scale text classification tasks without significant computational overhead.

Results for Baseline Model:

1. Training Results:

```
Training Accuracy: 0.8772127588510354
Training Micro F1 Score:0.8772127588510354

Training Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.93      0.89     12496
           1       0.91      0.82      0.87     11456

    accuracy                           0.88     23952
   macro avg       0.88      0.87      0.88     23952
weighted avg       0.88      0.88      0.88     23952


Training Confusion Matrix:
[[11585   911]
 [ 2030  9426]]
```

From these results, it can be noted that Random Forest scored quite good results as a baseline model, with training accuracy of 0.878, F1 score of 0.89 and 0.87 respectively on human written and AI generated labels with a macro average of 0.88.

Testing Gold dataset:

```
Testing Gold Accuracy: 0.8691935107376284
Testing Gold Micro F1 Score:0.8691935107376284
\Testing Gold Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.91      0.87     16272
           1       0.91      0.83      0.87     18000

    accuracy                           0.87     34272
   macro avg       0.87      0.87      0.87     34272
weighted avg       0.87      0.87      0.87     34272

\Testing Gold Confusion Matrix:
[[14772  1500]
 [ 2983 15017]]
```

By testing the gold dataset on the same baseline model, it resulted in a similar score as with the training dataset, justifying efficient and correct training. However, it can be figured out that false positives (1500) and false negatives (2983) were more in the testing dataset than training (911 and 2030) respectively.

Part 2

A) TRANSFORMER MODEL - DistilBert

Pre-Processing the Dataset

1. **Splitting the dataset:** The SemEval 2024 task 8 Subtask A monolingual (only English sources) dataset in present in jsonl format was imported into a dataframe and split into train (75%) and test sets (25%).
2. **Tokenization:** Text data was tokenized using **DistilBertTokenizer** function. This transforms the text data into numerical values that can be understood by the model. This was applied to train and test set in the form of batches.
3. **DataCollatorWithPadding:** This function is used to batch and pad the tokenized sequences. This ensures that all the sequences are the same length, helping in efficient processing by the model.

Fine-tuning the model

1. **Fine-tuning:** In this step, the pre-trained DistilBERT model was loaded and fine-tuned based on the training data using transfer learning. This updates the weight of model parameters to adapt it to the classification task at hand.

Model Evaluation

1. **Validation set:** The fine-tuned model was evaluated on validation set to assess its performance. Metrics such as Accuracy, macro-F1 and micro-F1 were calculated.
2. **Test Set (Gold Standard):** The model was also evaluated on the provided test set. It demonstrated an accuracy of 0.75, macro-f1 score of 0.74, and a micro-f1 score of 0.75. Figure 1 illustrates the classification report of the model.

{'0': {'precision': 0.8815458937198067, 'recall': 0.5607177974434612, 'f1-score': 0.6854481256103974, 'support': 16272.0}, '1': {'precision': 0.701195552
2113535, 'recall': 0.9318888888888889, 'f1-score': 0.8002480797671867, 'support': 18000.0}, 'accuracy': 0.7556605975723623, 'macro avg': {'precision': 0.
791370722965 5802, 'recall': 0.746303343166175, 'f1-score': 0.7428481026887921, 'support': 34272.0}, 'weighted avg': {'precision': 0.7868240757006612, 're
call': 0.7556605975723623, 'f1-score': 0.7457422191801397, 'support': 34272.0}}

Figure 1: Classification report for DistilBERT

Why DistilBERT?

As the name suggests, DistilBERT is a distilled version of BERT meaning it has fewer parameters and it is computationally efficient. It offers significant advantages in memory

usage and speed compared to models like BERT. Despite its smaller size, it offers competitive performance on various NLP tasks such as one at hand.

B) LLM Model - Beijing Academy of Artificial Intelligence

(For this model we have used the same kind of preprocessing apart from the PEFT and LoRA Configuration)

For the Generative AI with LLM Task we have used the the model as given on this link - https://huggingface.co/BAAI/bge-small-en-v1.5

This model is a small version of the BAAI BGE LLM.

USE OF PEFT:

We have also used a technique called PEFT to fine tune the LLM so that it can work on classification tasks. Traditionally LLMs are made for Natural Language Generation, and they require a sequence classification wrapper/module so that they can be optimized for classification. In this module, we have a pre-trained Transformer Model as a feature extractor and additional layers (classification heads) to perform classifications. This approach is known as Transfer Learning.

Because all the LLMs do not have modules like this, we found that a suitable way to train the BAAI/BGE LLM model was using Parameter Efficient Fine Tuning.

Parameter Efficient Fine Tuning (PEFT- https://huggingface.co/docs/peft/en/index):

This is a library which is useful for efficiently adapting Large Pretrained Models to downstream applications with fine-tuning only a small number of its parameters. This approach decreases the computational and storage costs which were very crucial in our setup.

For Parameter Efficient Fine Tuning – we used a library called LoRA.

LoRA - To make fine-tuning more efficient, LoRA's approach is to represent the weight updates with two smaller matrices (called update matrices) through low-rank decomposition.

```
    peft_config = LoraConfig(
        lora_alpha=lora_alpha,
        lora_dropout=lora_dropout,
        r=lora_r,
        bias="none",
        task_type=TaskType.SEQ_CLS,
        #task_type="CAUSAL_LM",
        target_modules=target_modules,
        modules_to_save=["score"]
    )
```

WHY BAAI's BGE Model?

The reasoning behind using this model was:

- Training Cost: Models like LLAMA require a lot of GPU Memory. We faced issues where we ran out of CUDA Memory while trying to train LLAMA models in both Kaggle (P100 – 16 GB) and Local (RTX 3070 – 8 GB) Gpu setup.
- Training Time: Training Time is huge because of the number of parameters. We tried reducing number of batches and also max_token_size to reduce training times. Even then it takes hours and sometimes runs out of memory scope after a certain amount of training.

  These are the reasons we chose BAAI's BGE LLM for our task.

Instructions on how to run the code:

- For all the notebooks for RF, DistilBert and BGE AAI LLM we can start running the code and then do the necessary imports.
- Commonly used libraries were:
- HuggingFace – Transformers, Dataset, Evaluate, Accelerate, Trainer, Training Arguments, Tokenizer TRL, BitsandBytes
- Other must have libraries - Pandas, Torch, CUDA environments, scikit-learn.

  When these common libraries are all installed, the code can be run when uploaded to Kaggle.
- *For Kaggle – please use a hardware accelerator (preferably GPU – P100), or the code might fail to run*
- *For Local systems – the local environment being run should support CUDA in Py torch.*
- The code should then probably run until the end and give a final JSON file.

Results and Discussion

Accuracy

Baseline (Random Forest)	**Model 1(DistilBERT)**	**Model 2(BAAI-BGE-LLM)**
0.869	0.755	0.803

Macro-F1

Baseline (Random Forest)	**Model 1(DistilBERT)**	**Model 2(BAAI-BGE-LLM)**
0.851	0.742	0.796

Micro-F1

Baseline (Random Forest)	Model 1(DistilBERT)	Model 2(BAAI-BGE-LLM)
0.869	0.755	0.803

It is evident from the above results that the random forest classifier demonstrates significantly better than the other two models. While constructing the baseline model, TF-IDF was used for feature engineering. Domain-specific knowledge may be incorporated into TF-IDF using n-grams, custom stop words, and other preprocessing methods. This may help with classification jobs where differentiating between classes is heavily reliant on terminology peculiar to a certain topic. Transformer-based models and generative LLMs, on the other hand, could have trouble adapting to new domains or would need a lot of fine-tuning to properly capture domain-specific subtleties. Additionally, TF-IDF exhibits computational efficiency, particularly in scenarios with huge datasets or limited hardware resources. Moreover, random forest classifier's better performance in your text classification challenge may be explained by several factors, including, effective feature engineering, and model complexity.

We also believe that our approach in PEFT (parameter efficient Fine tuning) which used only around 0.3% of the total parameters decreased it’s power in this task. We also used 50% of the provided train data to train our LLM, which otherwise would go out of CUDA_MEMORY. Although PEFT only tuned the relevant parameters, it still decreased the prediction capabilities, and we think GPT3 and GPT4 Models can better suit this task. If we are given the resources and time for it, it would be a fun experiment to train models with over a billion parameters (maybe in the Cloud or SageMaker). Also, LLMs are notoriously famous for not being able to generalize on provided data. Even our transformer model does not perform as well as tf-idf on Random forests because random forests are able to generalize better and handle huge datasets with limited resources. Without the restriction of limited resources, we believ that the other two models would have performed better.