Assignment 2

CSI 5386 – Natural Language Processing

Machine-Generated Text Detection

Students:

Kelvin Mock (300453668, kmock073@uOttawa.ca)

Jenifer Yu (300399089, wyu094@uOttawa.ca)

Sabrina Cai (300428657, hcai062@uOttawa.ca)

Professor: Dr. Diana Inkpen (diana.inkpen@uottawa.ca)

GitHub Repository: https://github.com/kmock930/Natural-Language-Processing.git

Contents

Part 1: Data Extraction & The Baseline

Introduction

The dataset is part of the SemEval 2024 Task 8 – Multidomain, Multimodel and Multilingual Machine-Generated Text Detection. It is downloaded from the Google drive link for Task A. With the help of the json library in Python, we extracted the data content from the file line-by-line and interpreted it in the format of a Pandas DataFrame for further modelling. Our end goal is to determine whether a piece of text is human-written or machine-generated, which forms a binary classification problem. Due to a limitation in the scope, we consider only English texts in the training data. Please run the Jupyter Notebook named part_1.ipynb sequentially to view end-to-end results.

Data Division

Our training set is composed of the complete text from "subtaskA_train_monolingual.jsonl". Our validation set is composed of the complete text from "subtaskA_dev_monolingual.jsonl". Our test set is composed of the complete set from "subtaskA_monolingual.jsonl". The training set and the validation set are labelled while the test set is not, and thus, only an "id" is attached corresponding to a piece of text.

Data Processing

To recognize those texts in a machine learning model, we need to vectorize them. As a result, our baseline model relies on the "TfidfVectorizer" which is from the scikit-learn library.

Modelling our Baseline Approach

Logistic Regression is applied as the baseline approach. We defined the maximum iterations to be 1000, which is the only customized parameter and otherwise by default.

Evaluation on the Baseline Model

With the use of the validation set, we evaluated the baseline model based on the accuracy, macro and micro F1 scores.

With the use of the unlabelled test set, our program generated the model's predictions into the file "Result_baseline.jsonl", which consists of objects with the "**id**" of a piece of text and the predicted **label** of the text.

The model is then saved so that we can perform a cross comparison with other models at a later stage.

Part 2: Deep Learning based Models

Introduction

Performing deep learning tasks is challenging given its huge demand on computational resources. However, we still believed its better performance. Hence, we implemented 2 deep learning-based models. One is using a pre-trained model, and another one is leveraging a recent generative LLM.

A Pre-Trained Model

Computational Requirements

With a challenge of GPU resources on my computer locally, we chose to train our model on a Linux-based virtual machine with the following specifications:

- GPU: RTX2080 Super
- vCPU: 8
- CPU Memory: 48GB
- GPU Memory: 8GB

Choices of Models

Given adequate resources, we intend to choose a pre-trained model in the BERT family, which is lightweight enough as a small project, from the HuggingFace platform. We considered the base model of BERT (i.e., bert-base-uncased), the DistilBERT model (i.e., distilbert-base-uncased), and RoBERTa (i.e., roberta-base). BERT is a classic go-to model which is widely supported and documented. However, it is a large model which incurs slower training and inferencing processes compared to other smaller models. Like the BERT model, RoBERTa excels at many benchmarks, given the same size of the BERT model, more data, and (proven) better set of hyperparameters. However, it slightly consumes more memory usage than BERT and is sometimes trained a bit slower than BERT. Additionally, it lacks the next-sentence-prediction head which typically is not critical for a classification problem. Given limited resources, albeit a potential performance boost, we eventually did not choose RoBERTa. Eventually, we chose the DistilBERT model. Although it is proven slightly less accurate than a full BERT or RoBERTa model in many tasks, it is also clearly documented with 40% fewer parameters than BERT, which gives a faster and more reasonable training time.

Technical Details of Our Choice

In this binary classification problem, the “TFDistilBertForSequenceClassification” is chosen given its classification head. A specific tokenizer is used – “DistilBertTokenizer” – which is different from the one that we used for the baseline model. It is then trained with Tensorflow in Python. If you have sufficient resources, please run the Jupyter Notebook named "part_2_deep_learning_training.ipynb".

The architecture of the DistilBERT model is summarized as follows:

Layer (type)	Output Shape	Param #
distilbert (TFDistilBertMainLayer)	multiple	66362880
pre_classifier (Dense)	multiple	590592
classifier (Dense)	multiple	1538
dropout_19 (Dropout)	multiple	0

Total params: 66,955,010

Trainable params: 66,955,010

Non-trainable params: 0

The model is fine-tuned in 5 epochs, with metrics shown below:

	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
Epoch 1	8.0683	0.4168	7.6246	0.5
Epoch 2	8.0672	0.4710	7.6246	0.5
Epoch 3	8.0672	0.4708	7.6246	0.5
Epoch 4	8.0672	0.4710	7.6246	0.5
Epoch 5	8.0672	0.4710	7.6246	0.5

Challenges and Observations

From the DistilBERT Model, where the core deep-learning-based fine-tuning lies in, we observed a few challenges:

- Unreasonably long training time
- Varying lengths from different texts in the dataset
- Incompatible model's saving method

Speaking of the prolonged training time, it causes much difficulty in debugging. Hence, we must treat the entire neural network as a black box where we make predictions from it by inferencing.

Each piece of texts from the dataset has varying lengths. While the model is being trained, those texts are converted into tensors. There exist inconsistent input shapes among those tensors. The inconsistencies led to incompatibility during prediction. Thus, we combined actual input texts with attention masks. In a transformer-based architecture, attention refers to the padding of empty spaces, which makes the sizes of tensors aligning with one another.

After the prolonged training process, for easier debugging, evaluation is separated to another notebook, and as a result, the trained model is saved to a directory and accessed via the notebook for evaluation. However, how we saved the model is the method for ordinary Tensorflow Keras models. Since this is a DistilBERT model from HuggingFace, the platform specifies another way to save a model, with intended input signatures. With the long training time, we decided not to retrain, but to adapt the traditional way of saving a deep-learning-based model. The challenge happens at the prediction phase, where we had to manually encode the input vectors again, such that they match the format which the model was trained for. We also manually implemented a for-loop to iterate over 5000 samples for the prediction. With the incompatible model's format, it causes hallucination where we noticed all labels being predicted on the same class.

Given all these challenges, the scores of this model are even lower than the baseline model.

A Zero-shot Classification Approach

In this section, we describe the implementation and evaluation of the **zero-shot classification** model, which utilizes the **facebook/bart-large-mnli** model from HuggingFace. This model was selected as an alternative to the fine-tuned DistilBERT, exploring the potential of large pre-trained language models to perform classification tasks without requiring task-specific training.

Model Selection and Motivation

For this approach, we chose the **facebook/bart-large-mnli** model because it has demonstrated strong capabilities in natural language inference (NLI) tasks and is particularly effective in zero-shot learning scenarios. Unlike traditional models that require fine-tuning on a specific task, this model can directly classify text into predefined categories based on its pre-trained knowledge. This allows us to classify machine-generated versus human-written text without the need for labeled data specific to this task.

The decision to use BART (Bidirectional and Auto-Regressive Transformers) was based on its ability to handle complex textual structures and its flexibility in zero-shot scenarios. By leveraging the pre-trained weights, we aimed to avoid the overhead associated with training a model from scratch, especially given the computational limitations.

Computational Challenges and Solutions

The implementation faced significant **computational constraints**, especially with respect to **GPU memory**. Initially, we attempted to run the model on a **NVIDIA GeForce (6GB VRAM)** GPU, but we encountered memory issues during the inference process. After several attempts to optimize GPU memory management, we switched to **CPU processing**, which provided more stable results.

Processing the full dataset of **34,272 test samples** required careful management of resources. To mitigate memory limitations, we truncated each text to **500 tokens**—the maximum context length supported by the model. This truncation helped avoid exceeding the model's context limits but led to the loss of some text information in longer samples. Despite this, the model was still able to produce reliable classifications.

Implementation and Processing Details

The implementation leveraged the **transformers** library's **pipeline** functionality for zero-shot classification. Here are the main steps in the implementation:

- **Text Preprocessing**: Each text sample was truncated to 500 tokens to align with the model's input limitations. This was necessary to prevent out-of-memory errors during processing, particularly when dealing with large datasets.
- **Classification**: We tested each piece of text against two candidate labels: "human-written" and "machine-generated". The model then classified each text by selecting the label with the higher probability based on its pre-trained knowledge.
- **Inference Mechanism**: The zero-shot classification approach uses the model's inference mechanism to evaluate the likelihood of each label. Predictions were made by comparing the probabilities for the two labels and selecting the higher one. This allowed us to classify text samples without the need for explicit training on labeled data.
- **Result Conversion**: The output from the model was then converted into a binary format, with **0 indicating human-written** and **1 indicating machine-generated**. The results were stored in a **JSONL file** format, which is consistent with the data format used by the baseline model.
- **Batch Processing**: Due to the computational intensity of the model, we processed the dataset in **batches** to prevent memory overload. Each sample took approximately **2.57 seconds** to process, and the entire dataset was processed sequentially. The batch processing ensured that we could handle large datasets without overwhelming the system's memory.

Evaluation and Results

The performance of the zero-shot classification model was evaluated on both the **validation set** and the **test set**. The following metrics were obtained:

- **Test Set Accuracy**: 50.85%
- **Macro F1 Score**: 43.58%
- **Micro F1 Score**: 50.85%

- **Validation Set Accuracy**: 52.06%

These results suggest that the zero-shot model is competitive, though not superior to the fine-tuned **DistilBERT** model. While zero-shot learning offers convenience and flexibility, the accuracy scores indicate that there may still be room for improvement, particularly in terms of fine-tuning the model for this specific task.

Challenges and Observations

Several **challenges** were encountered during the implementation of the zero-shot approach:

- **Memory Management**: Even with text truncation, the model required significant memory resources. This forced us to process the data on the CPU, which slowed down the overall inference time. A more powerful GPU could have alleviated some of these issues.
- **Processing Time**: Due to the large dataset and the time-consuming nature of processing each sample (approximately 2.57 seconds per sample), the total processing time for the full test set was approximately **24 hours**. This long processing time highlighted the importance of efficient model implementation and resource management.
- **Model Interpretability**: The zero-shot model's decision-making process remained somewhat opaque. It was difficult to understand why the model classified certain samples as machine-generated or human-written, as the results were based on pre-trained knowledge without insight into the underlying reasoning. This lack of interpretability is a common issue with deep learning models.
- **Memory Overflow**: Despite truncation, there were instances where the text exceeded the model’s context window, leading to potential information loss. This could have affected the model's accuracy, particularly when classifying longer texts.

Conclusion

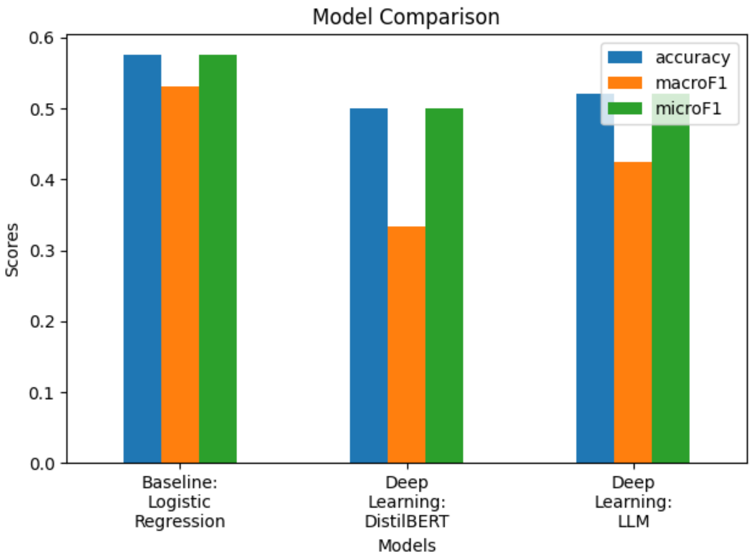
In conclusion, the zero-shot classification approach demonstrated competitive performance compared to the fine-tuned DistilBERT model, achieving a **test set accuracy of 50.85%** and a **macro F1 score of 43.58%**. The model's ability to perform machine-generated text detection without task-specific fine-tuning underscores the power of modern pre-trained language models. However, the performance gap between the zero-shot and fine-tuned models suggests that for some tasks, fine-tuning might still provide a performance boost.

This approach highlights the potential of zero-shot learning, especially in cases where computational resources for extensive training are limited. While the model is not perfect, its ability to perform well without fine-tuning makes it a valuable tool in scenarios with limited labeled data.

Results

From the validation set, we summarized the accuracies, macro and micro F1 scores for those 3 models (in %).

Models Metrics	Baseline **(Logistic Regression)**	Pre-trained **DistilBERT**	LLM **bart-large-mnli**
Accuracy (%)	57.54	50	52.06
Macro F1 (%)	53.16	33.33	43.58
Micro F1 (%)	57.54	50	50.85



From the test set, we exported a JSON file named "Result_distilBERT.jsonl", which contains results of predictions, with the "**id**" and **label**. For your convenience, you may find the script named "ModelSummarizer.py" helpful since it informs you the best model according to saved prediction arrays, and it also copies the corresponding model's result (.jsonl) file and renamed "Result_distilBERT.jsonl". As suggested in the plot above, our baseline model outperforms other models in our experiment.

Contribution

Name	Contributed Part	Details
Jenifer Yu	Part 1: Data Extraction & The Baseline	1. Downloaded and preprocessed the dataset. 2. Tokenized and vectorized texts. 3. Trained and evaluated the baseline model.
Kelvin Mock	Part 2: Deep Learning based Models	1. Refactored Jenifer's codes for better access to the dataset. 2. Load the pre-trained DistilBERT model from HuggingFace. 3. Set up a Linux-based virtual machine with sufficient computational resources. 4. Further trained the model. 5. Evaluated the model via a customized way for predictions, and adapted Jenifer's codes to show the results.
Sabrina Cai	Part 2: Deep Learning based Models	1. Implemented a zero-shot classification approach using the facebook/bart-large-mnli model. 2. Processed the complete test dataset (34,272 samples) to generate predictions 3. Evaluated the model using validation data (5,000 samples). 4. Generated and saved prediction files in JSONL format. 5. Created numpy arrays for model comparison visualization.