Midterm Project Report

Name: Goutham Deepak
Course: CSE 584
Date: 10/06/2024

Introduction

Problem statement:
With the ongoing technological rise, there is a rise of large language models, LLMs, such as GPT, Bloom, FLAN, etc. These advanced models have demonstrated their ability to generate creative and context aware text completions. In this project, we utilize these models to complete truncated sentences(x_i) and train a classifier that can predict the LLM responsible for generating the completion(x_j).

Objective:
There are two objectives:

1. Text generation: Utilize different LLMs to generate text completions for a data set of truncated sentences. Each model of LLM is expected to generate different and unique completions based on its mechanisms and its internal architecture.
2. Classification: Classification involves building a classifier that predicts which LLM generated the completion of a truncated sentence trained on the dataset that is compiled by running various LLMs for a given set of truncated sentences.

Dataset Curation:

Initial data curation: 2000 English sentences were taken and then truncated. So, this formed the xi for the project. This was done so that there is diversity in length, complexity and content. So the goal was to feed these into the LLMs and provide meaningful continuations (xj).

Selection of LLMs: For this task 5 LLMs where chosen. They were Flan-T5, BLOOM, GPT-2, CLTR and OPT.

- **Flan-T5**: It is a fine-tuned version of T5 known for its instruction-based task performance.
- **BLOOM**: It is a multilingual model trained to generate human-like text in multiple languages.
- **GPT-2**: It is a widely used model for text generation tasks known for coherent and fluent text.
- **CTRL**: It is a model designed to generate controlled text based on predefined control codes.
- **OPT**: It is a model from Meta that focuses on optimizing memory and compute for text generation tasks.

Completions generated by LLM: For each of the truncated sentence, completions were generated using all the five models. To maintain the quality of the completions, it was made sure that it truncated when it came across a ‘ . ’, ‘ ? ’ or a ‘ ! ’. This was done to ensure that only one sentence should be generated. Apart from that a retry mechanism was implemented

where so that a long enough sentence is generated. So the minimum word limit was kept as 10 and if a sentence isn't that long it tried again and again to generate a long sentence. An escape mechanism was kept where if it couldn't do it in 30 tries, it just took the small sentence generated. It was also made sure that there was a good amount of variety generated. The xi and the xj was combined together to form the full completed sentence for each of the LLM. Then after all of them were generated, certain LLMs couldn't generate more than 10 words for certain xi's even though a max length of 150 was given and it was prompted to do so. These rows were removed. Now a total of 1919 sentences where available and taken into consideration to be trained.

Dataset Structure:
Truncated_sentence: It contains the original truncated sentence.
Generated_xj: It contains the completion generated by the LLM.
Generated_Complete; It contains the combination of the truncated sentence and the completion given by the LLM.

Challenges in Data Curation and solutions:

- Ensuring quality of xi: Truncating the sentences was a challenge because if the sentence would be too short, it wouldn't make any sense, and if the sentence was too long, the LLMs generative capability wouldn't have been used completely. So we truncated, the sentences at a natural breaking point to make the sentence sensible, but incomplete thought.

- Hyperparameter tuning for LLM sentence generation: The dataset used has a wide variety of sentence types and styles. The sentences in the data set also had unique grammatical structure. This diversity was essential to test the generative capability of the LLMs being used. So different parameters were tested and taken into consideration. The same hyperparameters couldn't be taken to do a comparative analysis because of the decrease in performance. So, by trial and error, the optimal one was chosen.

Hypermeters to vary:

- **Max Length** (max_length): It defines how long the generated text can be. Initially, values between 100 and 150 tokens was taken. Finally, 150 into account.
- **Temperature** (temperature): It controls the randomness in text generation. Lower values like 0.7 made the text more deterministic, while higher values made it more random.
- **Top-k Sampling** (top_k): It limits the set of tokens from which the next word is selected. Lower values like 50 made the model to choose from a smaller set of high probability words.
- **Top-p Sampling** (top_p): It dynamically limits the token pool. Values between 0.9 and 0.95 provided a balance between coherence and diversity.

  Hyperparameters chosen:
  LLM-Specific Adjustments:
  Flan-T5:
  - Max Length: 150
  - Temperature: 0.7
  - Top-k: 50

Performance: Flan-T5 handled longer texts well and generated detailed completions with minimal adjustments. Lower temperatures kept the completions focused and relevant.

BLOOM:
- Max Length: 150
- Temperature: 0.9
- Top-p: 0.95

Performance: BLOOM was better with higher temperatures and top-p settings to increase the diversity of completions, as lower values made the text too repetitive.

GPT-2:
- Max Length: 150
- Temperature: 0.7
- Top-k: 50

Performance: GPT-2 provided fluent completions, but occasionally required retries for longer completions because it kept generating shorter continuations at lower temperatures.

CTRL:
- Max Length: 150
- Temperature: 0.95
- Top-k: 50

Performance: CTRL was tricky to manage. Higher temperatures were necessary to prevent the model from continuously repeating phrases.

OPT:
- Max Length: 150
- Temperature: 0.7
- Top-p: 0.9

Performance: OPT had similar characteristics to GPT-2 but sometimes generated redundant completions. Adjusting the top-p parameter helped balance randomness and relevance.

- Batch size optimization: Due to the different memory requirements, the batch size had to be altered so that the GPU can be used efficiently. For some models like T4, a batch size of 64 was used. But for the other, a batch size of 5 could only be used.

- Managing length and quality: When initially generated without any constraints, the sentences generated didn't make much sense or mainly were very short. Sentences were generated with either one word as the output or just with a full stop. Due to this additional tuning was done where the parameters had to be modified, and the 30-time repetition mechanism was taken into account. Out of 2000, 1919 xi values managed to get all the 5 xj's with more than 10 sentences. The remaining 81 were deleted. GPT and BLOOM especially struggled to generate long sentences initially.

- Memory constraint: Handling memory constraints was a concern since the models where large and to make it efficient, it has to be done in batches and there wasn't good GPU resources available. So the generation time took long very long or there were many run out of memory errors. So to solve this, used an external Nvidia RTX 4500 GPU and took base models. Also checkpoints where added so that after each batch it was saved.

  The models used were
  Flan-T5: **Version**: google/flan-t5-base.
  BLOOM: **Version**: bigscience/bloom-560m
  GPT-2: **Version**: gpt2
  CTRL: standard model
  OPT: **Version**: facebook/opt-1.3b

  OPT was very resource consuming as a bigger model was taken.

Experimental setup

A BERT based classifier was trained. It was done for the whole sentence and not just the xj so that it can maintain the conceptual understanding of the sentence. It was trained and evaluated under 2 different configurations:

Configuration 1: Training the classifier using all the 5 LLMs to predict which LLM generated which completion

Configuration 2: Excluding one LLM at a time from the training set to evaluate the impact of its absence on classification accuracy. This resulted in five separate experiments:

- Excluding BLOOM
- Excluding CTRL
- Excluding Flan-T5
- Excluding GPT-2
- Excluding OPT

Classifier

The classification task was performed using a BERT based deep learning model from Hugging Face's transformer library. BERT was chosen due to its efficient pre-trained contextual embeddings.

Architecture:

- BERT Encoder: The model processes the input sentences using the pre-trained BERT encoder, which outputs contextual embeddings of size 768.
- Dropout Layer: A Dropout layer with a rate of 0.5 was added to prevent overfitting by randomly dropping some of the neurons during training.
- Fully Connected Layer: The final layer is a fully connected layer that outputs logits for each of the 5 classes.

Loss Function: The model was trained using the standard CrossEntropyLoss.

Optimizer: The Adam optimizer was used with a learning rate of 2e-5. Adam was chosen because of its adaptive learning rate and efficiency.

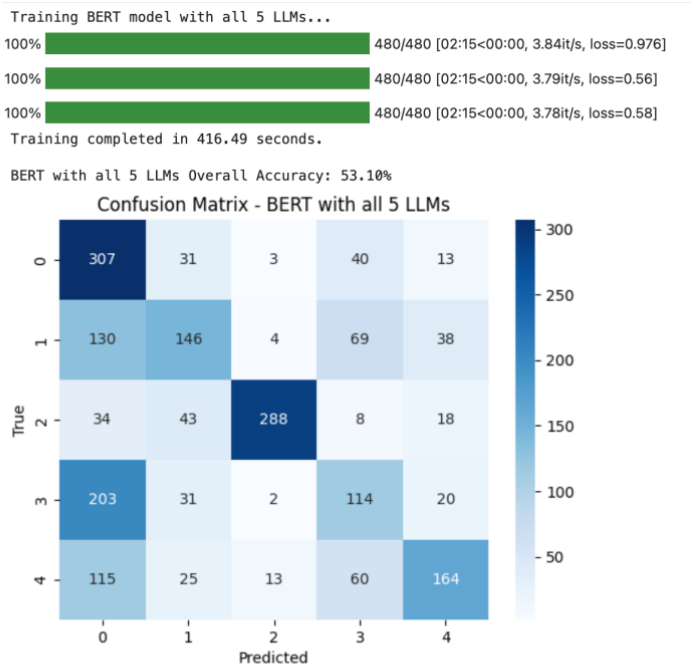
Epochs: The model was trained for 3 epochs, which provided sufficient time for the model to converge without overfitting.

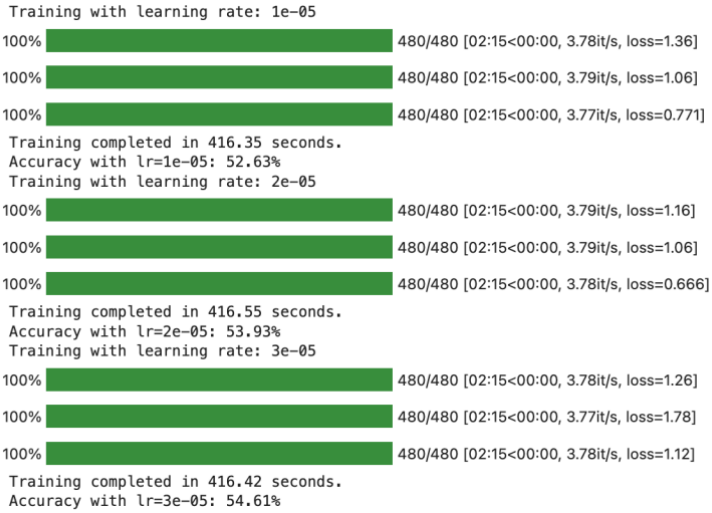
Batch Size: A batch size of 16 was used for both training and testing to balance memory usage and performance, as BERT requires significant memory for sentence embeddings.

The dataset was split into 80% for training and 20% for testing for each configuration to evaluate its generalization capabilities.

Results

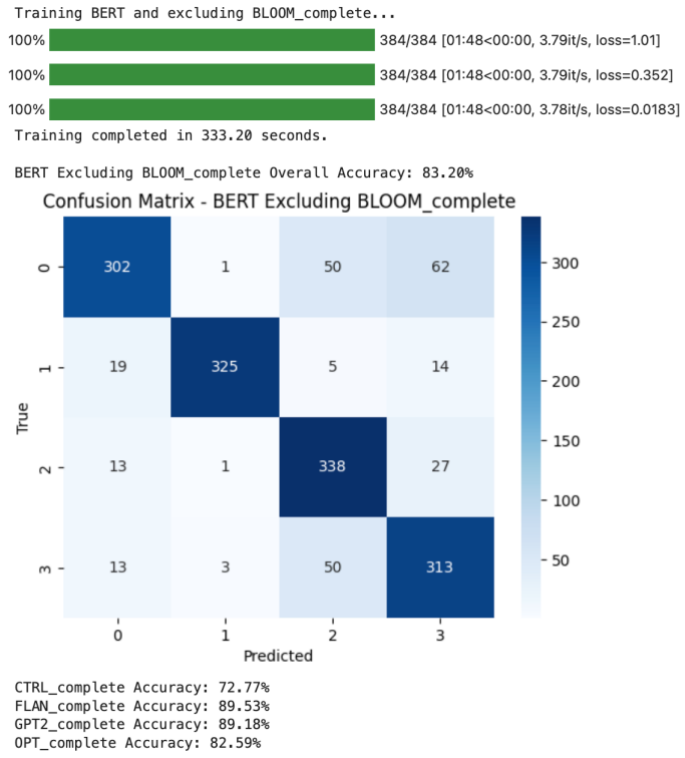
Configuration 1: The BERT model achieved an overall accuracy of **53.10%** when using all five LLM-generated completions (BLOOM, CTRL, FLAN, GPT-2, and OPT).





Three different learning rates (1e-5, 2e-5, and 3e-5) were tested to optimize the BERT model. The best accuracy achieved was 54.61% with a learning rate of 3e-5 during the tuning phase.

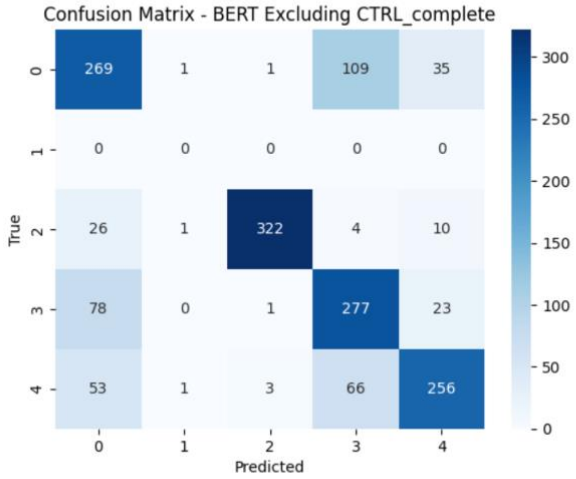
Configuration 2: To assess the impact of each LLM on the overall performance on LLM at a time was excluded and the BERT model was retrained.



Removing BLOOM significantly improves overall accuracy. This indicates that BLOOM may introduce noise or variability that affects the model's classification performance.

```
Training BERT and excluding CTRL_complete...
100% 384/384 [01:48<00:00, 3.78it/s, loss=0.356]
100% 384/384 [01:48<00:00, 3.78it/s, loss=0.382]
100% 384/384 [01:48<00:00, 3.79it/s, loss=0.0337]
Training completed in 333.28 seconds.

BERT Excluding CTRL_complete Overall Accuracy: 73.18%
```

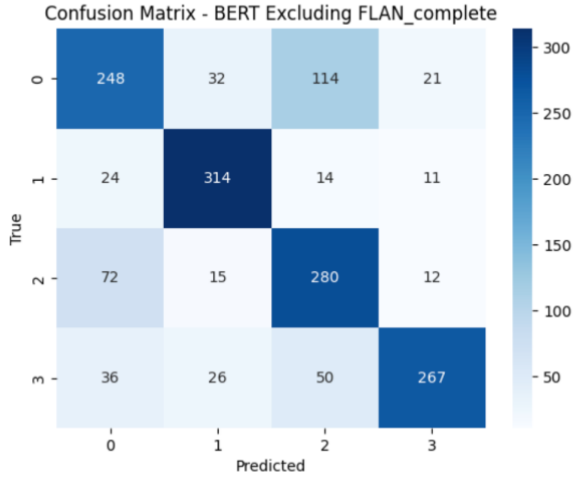



```
BLOOM_complete Accuracy: 64.82%
CTRL_complete Accuracy: nan%
FLAN_complete Accuracy: 88.71%
GPT2_complete Accuracy: 73.09%
OPT_complete Accuracy: 67.55%
```

Excluding CTRL shows a moderate improvement. This suggests that while CTRL's impact isn't as substantial as BLOOM's.

```
Training BERT and excluding FLAN_complete...
100% 384/384 [01:48<00:00, 3.79it/s, loss=0.145]
100% 384/384 [01:48<00:00, 3.77it/s, loss=0.0519]
100% 384/384 [01:47<00:00, 3.82it/s, loss=0.00843]
Training completed in 333.14 seconds.

BERT Excluding FLAN_complete Overall Accuracy: 72.20%
```

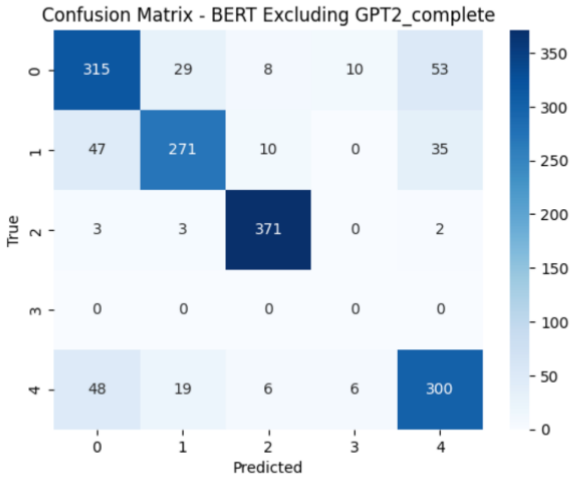



```
BLOOM_complete Accuracy: 59.76%
CTRL_complete Accuracy: 86.50%
GPT2_complete Accuracy: 73.88%
OPT_complete Accuracy: 70.45%
```

Excluding FLAN resulted in a slight decrease in accuracy. This confirming that FLAN contributes significantly to the model's classification performance.

```
Training BERT and excluding GPT2_complete...
100% 384/384 [01:48<00:00, 3.80it/s, loss=0.0011]
100% 384/384 [01:48<00:00, 3.79it/s, loss=0.127]
100% 384/384 [01:48<00:00, 3.77it/s, loss=0.00118]
Training completed in 333.19 seconds.

BERT Excluding GPT2_complete Overall Accuracy: 81.84%
```

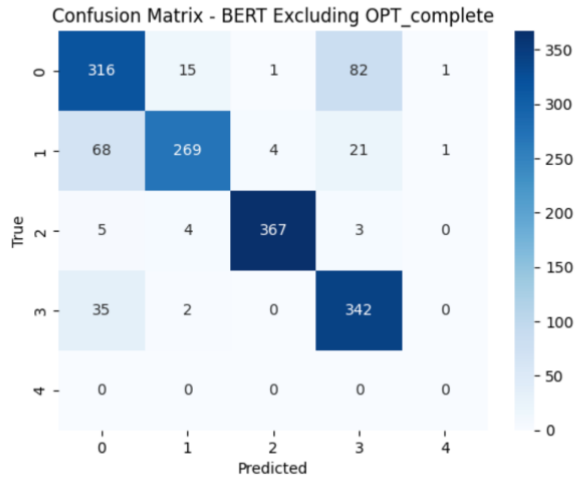



```
BLOOM_complete Accuracy: 75.90%
CTRL_complete Accuracy: 74.66%
FLAN_complete Accuracy: 97.89%
GPT2_complete Accuracy: nan%
OPT_complete Accuracy: 79.16%
```

Removing GPT-2 increased the accuracy substantially. This suggesting that GPT-2's lower accuracy has a negative effect when included in the model.

```
Training BERT and excluding OPT_complete...
100% 384/384 [01:48<00:00, 3.77it/s, loss=0.00605]
100% 384/384 [01:48<00:00, 3.79it/s, loss=0.00113]
100% 384/384 [01:48<00:00, 3.78it/s, loss=0.0982]
Training completed in 333.31 seconds.

BERT Excluding OPT_complete Overall Accuracy: 84.24%
```




```
BLOOM_complete Accuracy: 76.14%
CTRL_complete Accuracy: 74.10%
FLAN_complete Accuracy: 96.83%
GPT2_complete Accuracy: 90.24%
OPT_complete Accuracy: nan%
```

Similar to GPT-2, excluding OPT also increases the overall accuracy, confirming its lesser contribution to the model's classification ability.

The results show that BLOOM and FLAN are the most reliable LLMs when used for text classification and they significantly contributing to the model's accuracy. On the other hand, GPT-2 and OPT tend to hinder overall performance when included.

Related work

2 base paper were chosen and gone through before doing this project.

Paper 1: Smart Expert System: Large Language Models as Text Classifiers
https://arxiv.org/pdf/2405.10523
This paper investigates the use of Large Language Models (LLMs) as intelligent expert systems capable of performing text classification tasks. It used the generative capability of LLMs to classify textual data. Different LLMs are evaluated as accessed based on the accuracy.

Methodology:

- The authors used different LLMs (e.g., GPT, BERT-based models) and fine-tune them for text classification tasks.
- Different domain datasets are used. Eg. medical text, legal documents, and news articles. This is done to see how general the models will perform
- The paper compares traditional machine learning models like CNNs and LSTMs with transformer-based models such as BERT and GPT. It highlights that pre-trained LLMs can handle complex language patterns more efficiently. So it will surpass the traditional models.

Challenges and Limitations: According to the paper while LLMs can perform well in text classification, they get biased towards frequently seen patterns in training data. So because of this the accuracy will reduce.

Solution: The authors suggest that combining LLMs with meta-learning approaches might help to overcome this.

Paper Relevance:
This paper is aligned with the project as it investigates using LLMs for text classification.

- The challenges discussed in the paper are the same challenges faced.
- The same technique of prompt engineering is used.
- The bias between different models is discussed. It is important as different LLM completions like GPT-2, BLOOM, CTRL are used which may generate similar outputs.

Paper 2: Large Language Model (LLM) AI text generation detection based on transformer deep learning algorithm
https://arxiv.org/pdf/2405.06652
This paper presents a method for detecting AI-generated text using a deep learning model based on the Transformer architecture. It aims to improve the accuracy of identifying whether text is AI-generated. The authors combine various neural network layers like LSTM, Transformer, and CNN to classify and label sequences effectively.

Methodology:

- A deep learning model with multiple layers—such as embedding, LSTM, TransformerBlock, Conv1D, and GlobalMaxPooling1D—is built. It captures both long-

term dependencies (using LSTM and Transformer layers) and local features (using CNN layers).
- The training process uses a combination of optimization techniques like Adam and early stopping to ensure the model achieves the best accuracy without overfitting.
- The model is validated using an open-source dataset consisting of AI-generated and human-written texts.

Relevance:
- This paper's approach highlights the use of Transformer architectures for text classification tasks, similar to how LLMs are fine-tuned to classify text across different domains.
- By adapting such architectures for detecting whether text is AI-generated, the model aligns with techniques used for sentence classification in various LLMs.
- Integrating CNN and LSTM layers alongside the Transformer enables the model to capture complex patterns in sequences, making it relevant when considering the structural analysis needed for differentiating between outputs of different LLMs.

So by considering the strategies used in these paper, different methods have been implemented which was essential in improving the models performance and also determining which steps to take.