CSE584: Machine Learning
On Classifying LLMs Generated Texts.

Darshan Chudiwal

October 6, 2024

Problem Statement

Given a set of truncated texts, for each piece of text xi, such as "Yesterday I went", ask different Large Language Models (LLMs) to complete it by appending xj ="to Costco and purchased a floor cleaner." so you get a complete text like "Yesterday I went to Costco and purchased a floor cleaner." from each LLM. The same xi leads to different xj. The goal is to build a deep learning classifier to figure out, for each input (xi, xj), which LLM was used for this pair.

Dataset Curation

The dataset is created in three steps.

- The xi's are extracted from real world text. The Brown corpora are used to extract truncated sentences from texts that are classified into different genres like 'adventure', belles lettres, editorial, 'fiction', 'government', 'hobbies', 'humor', 'learned', lore, 'mystery', 'news', religion, reviews, romance, and 'science fiction'. The idea is to obtain a wide variety of truncated sentences that reflect the diversity of linguistic structures and vocabularies across these genres. Further, an equal number of sentences is chosen from each genre.
- Next, the extracted xi's are sent as prompts to different LLMs using the **hugging face** transformers library. We choose five distinct LLMs as follows – 'openai-community/gpt2', 'google/gemma-2-2b-it', 'meta-llama/Llama-3.2-1B','microsoft/Phi-3.5-mini-instruct', 'Qwen/Qwen2.5-1.5B-Instruct'. Each model generates 3 completions for one xi with the num_return_sequences parameter set to 3. This allows for a better exploration of a model's variability in responses to the same prompt. The top_k is set to 50 for a good balance between creativity in word choices and not becoming too random to keep the context relevant. In addition, a max length of 50 is set for the generation.
- Finally, the first sentence from a generated completion is chosen as the xj since the first sentence is the most coherent and contextually relevant sentence.

The dataset contains 22,500 samples. The dataset features are as follows:

1. truncated_sentence (Text): Contains truncated sentences of up to 3 to 4 words extracted from the Brown Corpus.
2. generated_sentence (Text): Generated completion for every xi.
3. model (Text): The model that generated the completion.
4. genre (Text): The genre of the truncated_sentence. Though not of much use given the length of the prompts (3 to 4 words), it could make for interesting analysis in the future.

Methodology : Bert based classifier

BERT transformer

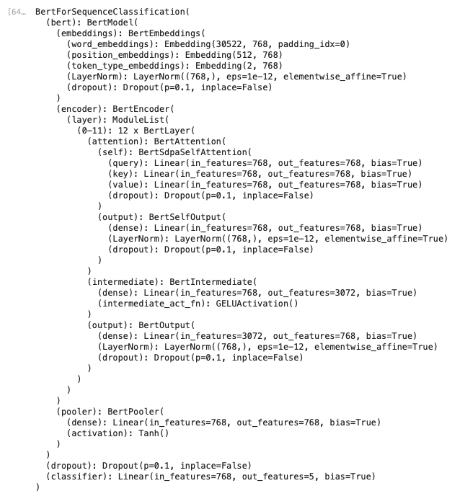
Bert was made to capture the bidirectional context of a sentence, which essentially means that it can understand the meaning of the words that precede it as well as the words that proceed it[1]. This makes BERT well suited for tasks that demand nuance as in this case, the context from the prompt and completion are crucial to accurately identifying the model that generated the text.

Furthermore, BERT has been pre-trained on massive corpora, making it capable of general language representations[1].

BERT also demonstrates a state-of-the-art performance in various classification tasks like sentiment analysis and natural lanugage inference[2].

Model architecture

The xi and xj are concatenated like so- "{truncated_sentence} [SEP] {completion}." This concatenated sentence is then tokenized using the BertTokenizer class and put in a custom Dataset class (from pytorch) for efficient batching and processing. The model is a classification model based on BERT, where the [CLS] token representation from the BERT encoder is used to predict which model



```
[64…  BertForSequenceClassification(
        (bert): BertModel(
          (embeddings): BertEmbeddings(
            (word_embeddings): Embedding(30522, 768, padding_idx=0)
            (position_embeddings): Embedding(512, 768)
            (token_type_embeddings): Embedding(2, 768)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (encoder): BertEncoder(
            (layer): ModuleList(
              (0-11): 12 x BertLayer(
                (attention): BertAttention(
                  (self): BertSdpaSelfAttention(
                    (query): Linear(in_features=768, out_features=768, bias=True)
                    (key): Linear(in_features=768, out_features=768, bias=True)
                    (value): Linear(in_features=768, out_features=768, bias=True)
                    (dropout): Dropout(p=0.1, inplace=False)
                  )
                  (output): BertSelfOutput(
                    (dense): Linear(in_features=768, out_features=768, bias=True)
                    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                    (dropout): Dropout(p=0.1, inplace=False)
                  )
                )
                (intermediate): BertIntermediate(
                  (dense): Linear(in_features=768, out_features=3072, bias=True)
                  (intermediate_act_fn): GELUActivation()
                )
                (output): BertOutput(
                  (dense): Linear(in_features=3072, out_features=768, bias=True)
                  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                  (dropout): Dropout(p=0.1, inplace=False)
                )
              )
            )
          )
          (pooler): BertPooler(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (activation): Tanh()
          )
        )
        (dropout): Dropout(p=0.1, inplace=False)
        (classifier): Linear(in_features=768, out_features=5, bias=True)
      )
```

Figure 1: *Model Architecture*

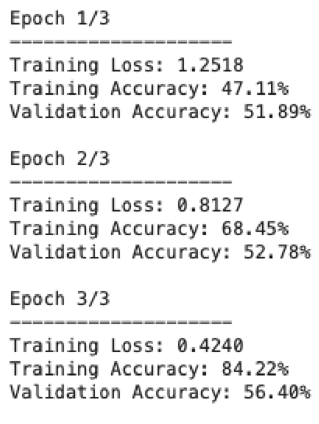
generated the completion. After encoding the concatenated sequence, the [CLS] token embedding is passed through a fully connected layer (classifier), which outputs logits corresponding to the different models. The model is trained using the cross-entropy loss function.

The AdamW lr is set to 3e-5. We run the model for a total of 3 epochs.

The BertForSequenceClassification can be described as seen in Figure 1.

Results

We start off with a Dual BERT classifier where the prompt and the completion are tokenized separately before being concatenated together. The model achieves a validation accuracy of 55%. Then, we try another approach where the two sentences are concatenated together separated by a [SEP] tag and tokenized before getting plugged in the model.



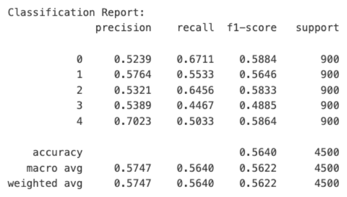
```
Epoch 1/3
--------------------
Training Loss: 1.2518
Training Accuracy: 47.11%
Validation Accuracy: 51.89%

Epoch 2/3
--------------------
Training Loss: 0.8127
Training Accuracy: 68.45%
Validation Accuracy: 52.78%

Epoch 3/3
--------------------
Training Loss: 0.4240
Training Accuracy: 84.22%
Validation Accuracy: 56.40%
```

Figure 2: *Accuracy over epochs*

The model gives a validation accuracy of 56.4% shown in Figure 2.



```
Classification Report:
              precision    recall  f1-score   support

           0     0.5239    0.6711    0.5884       900
           1     0.5764    0.5533    0.5646       900
           2     0.5321    0.6456    0.5833       900
           3     0.5389    0.4467    0.4885       900
           4     0.7023    0.5033    0.5864       900

    accuracy                         0.5640      4500
   macro avg     0.5747    0.5640    0.5622      4500
weighted avg     0.5747    0.5640    0.5622      4500
```

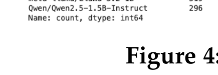
Figure 3: *Classification Report*

The classification report is shown in Figure 3.

Class 0 (Qwen/Qwen2.5-1.5B-Instruct) shows the highest recall at 0.6711, suggesting that the model is able to capture a large portion of the actual class 0 instances. However, the precision is quite low at 0.5239, meaning a significant portion of its predictions are false positives, which lowers its F1-score to 0.5884. Class 1 (google/gemma-2-2b-it) has relatively balanced performance with a precision of 0.5764, recall of 0.5533, and an F1-score of 0.5646, though none of these metrics stand out in particular. Class 2 (meta-llama/Llama-3.2-1B) is another area of concern with lower precision at 0.5321 and recall of 0.6456, yielding an F1-score of 0.5833. While the recall is decent, the precision being low implies that the model is misclassifying other instances as class 2 frequently. Class 3 (microsoft/Phi-3.5-mini-instruct) struggles the most, with the lowest recall of 0.4467 and a relatively low precision of 0.5389, resulting in an F1-score of 0.4885. This indicates that the model has significant difficulty correctly identifying class 3, both in terms of capturing actual instances and avoiding misclassifications. Class 4 (openai-community/gpt2) stands out with the highest precision at 0.7023, which means the model is good at minimizing false positives for this class. However, its recall is 0.5033, meaning half of the actual instances of class 4 are not being captured by the model. The F1-score for class 4 is 0.5864, showing that while precision is high, there's a trade-off with recall.

Experiments and Analysis

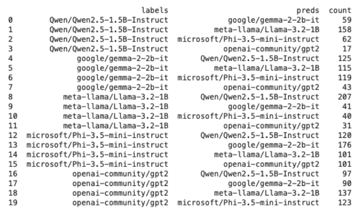
The results are further analysed to identify patterns if any particular model was confused for someone else. The incorrect guesses per class are shown in Figure 4. The Qwen/Qwen2.5-1.5B-Instruct has the least number of incorrect guesses implying the model understands the model's style the best while the model performs the worst with microsoft/Phi-3.5-mini-instruct model which has the most parameters of the five.



```
[71…  labels
      microsoft/Phi-3.5-mini-instruct    498
      openai-community/gpt2              447
      google/gemma-2-2b-it               402
      meta-llama/Llama-3.2-1B            319
      Qwen/Qwen2.5-1.5B-Instruct         296
      Name: count, dtype: int64
```

Figure 4: *Incorrect Guesses Per Class*

Class mismatch is another area that is looked at to identify which models were mistaken for which ones and what was the frequency of this. The Figure. 5 shows all the pairs and the frequency of the mismatches.



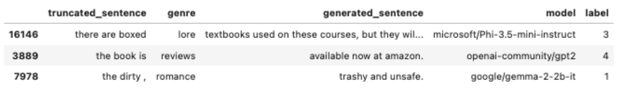
```
                          labels                            preds  count
0         Qwen/Qwen2.5-1.5B-Instruct           google/gemma-2-2b-it     59
1         Qwen/Qwen2.5-1.5B-Instruct          meta-llama/Llama-3.2-1B    158
2         Qwen/Qwen2.5-1.5B-Instruct  microsoft/Phi-3.5-mini-instruct     62
3         Qwen/Qwen2.5-1.5B-Instruct           openai-community/gpt2     17
4               google/gemma-2-2b-it       Qwen/Qwen2.5-1.5B-Instruct    125
5               google/gemma-2-2b-it          meta-llama/Llama-3.2-1B    115
6               google/gemma-2-2b-it  microsoft/Phi-3.5-mini-instruct    119
7               google/gemma-2-2b-it           openai-community/gpt2     43
8            meta-llama/Llama-3.2-1B       Qwen/Qwen2.5-1.5B-Instruct    207
9            meta-llama/Llama-3.2-1B           google/gemma-2-2b-it     41
10           meta-llama/Llama-3.2-1B  microsoft/Phi-3.5-mini-instruct     40
11           meta-llama/Llama-3.2-1B           openai-community/gpt2     31
12   microsoft/Phi-3.5-mini-instruct       Qwen/Qwen2.5-1.5B-Instruct    120
13   microsoft/Phi-3.5-mini-instruct           google/gemma-2-2b-it    176
14   microsoft/Phi-3.5-mini-instruct          meta-llama/Llama-3.2-1B    101
15   microsoft/Phi-3.5-mini-instruct           openai-community/gpt2    101
16             openai-community/gpt2       Qwen/Qwen2.5-1.5B-Instruct     97
17             openai-community/gpt2           google/gemma-2-2b-it     90
18             openai-community/gpt2          meta-llama/Llama-3.2-1B    137
19             openai-community/gpt2  microsoft/Phi-3.5-mini-instruct    123
```

Figure 5: *Class Mismatch Pairs*

The Qwen/Qwen2.5-1.5B-Instruct is most often misunderstood for meta-llama/Llama-3.2-1B. The inverse of this is also found to be true where meta-llama/Llama-3.2-1B is misunderstood for Qwen/Qwen2.5-1.5B-Instruct. Perhaps, the similar sizes of these relatively small LLMs factors into this (1B parameters for Llama and 1.5B parameters for Qwen). The microsoft/Phi-3.5-mini-instruct is largely misclassified as google/gemma-2-2b-it. GPT2 is mistaken for meta-llama/Llama-3.2-1B and microsoft/Phi-3.5-mini-instruct. The google/gemma-2-2b-it model is the most ambiguous considering a similar number of misclassifications into Owen/Qwen2.5-1.5B-Instruct, meta-llama/Llama-3.2-18 and microsoft/Phi-3.5-mini-instruct.

Finally, we run the model through LIME (local interpretable model-agnostic explanations) for some datapoints from the validation set to analyze how the model makes certain decisions for better interpretability.

We use LIME for the datapoints seen in Fig. 9.



	truncated_sentence	genre	generated_sentence	model	label
16146	there are boxed	lore	textbooks used on these courses, but they wil...	microsoft/Phi-3.5-mini-instruct	3
3889	the book is	reviews	available now at amazon.	openai-community/gpt2	4
7978	the dirty ,	romance	trashy and unsafe.	google/gemma-2-2b-it	1

Figure 6: *Datapoints for LIME analysis*

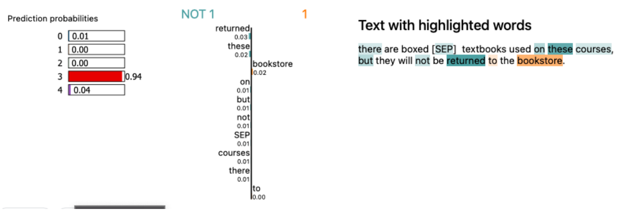


Figure 7: *LIME analysis for row 1*

The bar chart on the left shows the model's predicted probabilities for different classes. On the right side of the prediction probabilities, there is a list of words (tokens) with corresponding weights (small numbers next to the words). These weights show how much each word contributes to the prediction. For example, in the first image, the word "returned" has a weight of 0.03, meaning it slightly influences the prediction. Conversely, words with smaller weights (close to 0) contribute less. Unfortunately the model agnostic nature of LIME leads it to incorrectly interpret the [SEP] as a part of the corpus.

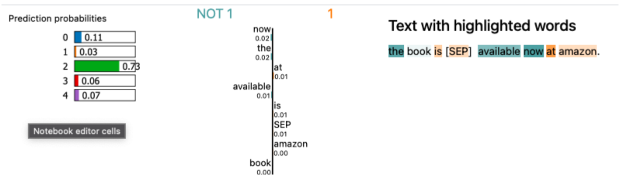


Figure 8: *LIME analysis for row 2*

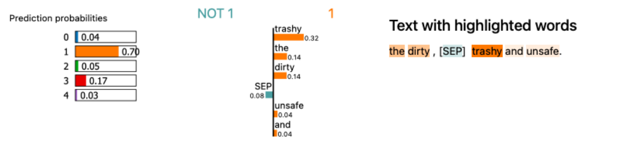


Figure 9: *LIME analysis for row 3*

The text on the far right is color-coded, highlighting words in different colors. Words that strongly contribute to the prediction are more prominently highlighted.n the third image, the word "trashy" is highlighted in orange and has the highest weight (0.32), meaning it is the most important word for the model's prediction of class '1'.

Related Works

In recent advancements in detecting machine-generated text, Chen et al. (2023) propose an innovative approach by reframing the task of LLM identification as a next-token prediction problem using the T5 model [3]. Their method, termed T5-Sentinel, eliminates the need for an additional classification layer, leveraging T5's inherent token prediction capabilities for implicit classification. This approach is evaluated using the OpenLLMText dataset, which includes text generated by multiple LLMs such as GPT-3.5, PaLM, and LLaMA. The results demonstrate superior performance compared to existing classifiers, including OpenAI's text classifier and ZeroGPT. The interpretability studies also reveal the model's capability to distinguish subtle stylistic differences between human and machine-generated text, making it a highly effective solution for LLM detection in large-scale datasets.

Abburi et al. (2023) explore the classification of generative AI text by utilizing an ensemble approach combining the strengths of various pre-trained large language models (LLMs) [4]. Their work focuses on two tasks: distinguishing AI-generated text from human-written text and attributing the AI-generated text to a specific language model. By leveraging multiple LLMs and combining the output probabilities with traditional machine learning classifiers, their approach demonstrates superior performance in both English and Spanish datasets. This method ranked first in model attribution, showcasing its robustness in detecting the origin of AI-generated content. Their findings align with other efforts to enhance the reliability of detecting AI-generated text while improving the attribution of the models involved(2).

Mo et al. (2023) introduce a transformer-based model for detecting AI-generated text with high accuracy [5]. Their approach integrates various deep learning techniques such as LSTM, Transformer, and CNN layers for sequence classification. By preprocessing text through normalization and employing a hybrid model architecture, the authors achieved 99% accuracy in classifying AI-generated content, demonstrating the model's robustness in detecting such texts. This work contributes significantly to the domain of AI text detection by enhancing accuracy and scalability, offering valuable insights for future advancements in this area(3).

Wu et al. (2024) [6] provide a comprehensive survey on methods for detecting LLM-generated text, emphasizing the growing necessity of robust detectors due to the rapid adoption of LLMs in daily life. The paper discusses detection methods, such as watermarking, statistical, and neural-based approaches, highlighting their strengths and weaknesses. It also explores challenges in detection, including out-of-distribution problems and real-world data issues, and suggests future research directions to improve LLM-generated text detection accuracy and reliability.

Conclusion and Future Work

While the model gives a subpar performance, the work highlights a myriad of things to be changed for a better understanding of AI text. A brief prompt might've helped in getting data that was more context focused. The attempt here was to let LLM do all the "thinking" without a prompt that might have led to more random responses.

Further, the model here was a fine-tuned BERT model that could've been used in a different sort of set-up.

References

[1] Jacob Devlin. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[2] Yinhan Liu. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[3] Yutian Chen et al. "Token prediction as implicit classification to identify LLM-generated text". In: *arXiv preprint arXiv:2311.08723* (2023).

[4] Harika Abburi et al. "Generative ai text classification using ensemble llm approaches". In: *arXiv preprint arXiv:2309.07755* (2023).

[5] Yuhong Mo et al. "Large language model (llm) ai text generation detection based on transformer deep learning algorithm". In: *arXiv preprint arXiv:2405.06652* (2024).

[6] Junchao Wu et al. "A survey on llm-gernerated text detection: Necessity, methods, and future directions". In: *arXiv preprint arXiv:2310.14724* (2023).