On Building an LSTM for LLM-Generated Text Classification

Kade E. Carlson

October 6, 2024

1 Introduction

Large languages models (LLMs) have become center stage in the past few years due to the rise of agents like ChatGPT and its ilk. LLMs have shown unparalleled utility, however, much concern has arisen due to the uses of LLMs in education, writing, art, and other pursuits. With that being said, it has become ever more important to detect when AI-generated text is being used and in what context. Since ChatGPT, many other LLMs have been developed such as Mistral 7B and open source models like LLaMa 2.

The increase in more options of LLMs that all behave in different ways and can be fine-tuned to meet other needs may make it more difficult to detect when LLM-generated text is being used. This work will focus on developing a deep learning model that can classify what LLM generated a given string of text. The LLMs used are OpenAI's GPT-4o-mini, Claude 2.1, Commander, Jamba 1.5, Mistral 7B, LLaMa 3.2 1B, and GPTNeo 1.3B.

2 Related Work

A lot of research efforts have been focused on developing models for detecting when text has been AI generated. Recently developed was DetectGPT [2]. DetectGPT is unique in the sense that it is not a machine learning model and therefore requires no training, no data, and no extra fine tuning. DetectGPT exploits a simple characteristic of LLM text generation. This characteristic is that most LLMs sample from negative curvature region's of the models log likelihood function. DetectGPT takes text samples and then applies a perturbation to the sample. If the difference in the perturbation is large then the text sample is likely to be generated by an LLM. DetectGPT is highly accurate at detecting generated text

In 2019, OpenAI released GPT-2 and later did research on detecting differences in human versus machine generated text [3]. To do this they fine-tuned two large language models based on Google's BERT. The work of OpenAI found that longer texts increase accuracy where shorter texts were much more difficult to classify. Models trained on longer text samples achieved close to 90% accuracy where shorter texts only had 50-60%.

[1] proposed several different types of models for binary classification between humand and machine generated text. These included convolutional neural networks, bag-of-words methods, and fine-tuned models such as Google's BERT to detect deep fake tweets. Most of these methods achieved between 80% and 90% accuracy.

Many of the previous works discussed focus on developing classifiers for machine vs human generated text. However, [4] worked on developing classifiers for determining what text-generation model developed a certain text, similar to this work. [4] used five different classifiers to determine artifacts of machine generated text. These classifiers are a linear Bag of Words model, a nonlinear Bag of Words model with a ReLU activation layer, a 1D convolutional neural network with softmax activation, an LSTM with softmax activation, and a four layer transformer encoder. [4] compared the different classifiers at how well they discovered artifacts in machine generated text.

To the author's knowledge there is not previous work done in building a classifier that can determine what machine generated text given a set of different machines to choose from. The contributions of this work are an LSTM classifier that can predict with 40%-50% accuracy what LLM generated a prompt of text. Another contribution of this work is a discussion on how different prompting techniques can help the accuracy of the classifier by manipulating the behavior of the LLMs.

3 Dataset Curation

The prompts that are being fed to the LSTM have a requirement that the LLM must finish a prompt given to it. For example, an input prompt to an LLM would be "Yesterday I went" and the LLM would finish the prompt saying "Yesterday I went to Costco and bought a floor cleaner". There is currently no known dataset like this so the dataset had to be generated by the author. Prompts were chosen to create a diverse set of completion prompts from the LLMs. Prompts were created to aim for the LLMs to express emotions like sentiment or happiness, reasoning ability, humor, and potential personality traits like favorite foods or songs. The prompts used to build the training set are the following:

1. "Today I went"
2. "Yesterday I was feeling"
3. "I had an idea about"
4. "I played a game"
5. "I heard this fact the other day:"

6. "The best meal I ever had was"
7. "A strange dream I had involved"
8. "My inspiration comes from"
9. "Technology has shaped us in these ways"
10. "If I could meet any historical figure it would be"
11. "Great hobbies include"
12. "My favorite song is"
13. "My favorite food is"
14. "My daily routine involves"
15. "I like this t-shirt because"
16. "My biggest fear is"
17. "I find fault in"
18. "I like math because"
19. "12 times 12 is 144. I think about it this way"
20. "Listen to this one-liner joke"

These are 20 prompts and are considered the "zero-shot" dataset. Since this paper looks to experiment on how prompting affects the LSTM classifier, another dataset was created called the "few-shot" dataset. It uses the exact same prompts as the zero-shot version but provides an example to the LLM for each prompt on what the author is looking for. The LLMs chosen were:

1. GPT-4o-mini
2. Command r+
3. Claude 2.1
4. Jamba 1.5 Large
5. LLaMa 3.2 1B
6. Mistral Large
7. GPT-NEO 1.3B

Many of these had APIs that were callable and can receive responses. Some of the others are open source and require a little more set up such as LLaMa 3.2B. Also, many of the models used that didn't have APIs had to be smaller parameter size due to memory constraints on the author's hardware (RTX 3070 GPU). Each LLM was queried 100 times for each prompt, giving 14,000 training points for the dataset. Each prompt also has a random temperature and max number of tokens between the ranges of 0 and 1 and 30 and 100 respectively. The test set had 5 prompts and 25 queries each prompt for each LLM giving 875 test points.

4 LSTM Classifier

An LSTM was chosen due to its ability to handle text data very well and not requiring a significant amount of testing data. The model was implemented in PyTorch. Before training the model, some data prepocessing had to be done. Preprocessing of the data included combining the input prompt and the completion of the prompt into one prompt. Then the models had to be mapped to integers so that the model could be trained properly. After this, the prompts were passed through the BERT Tokenizer so that the model could understand the language in numerical terms. Finally, the training dataset was split 80% training data and 80% validation data.

4.1 Model Architecture

The model architecture consists of the embedding layer for vectorizing the input words. This also helps capture words that are semantically similar for more effective training. Next is the LSTM layer that has a hidden dimension. This layer is repeated twice. Finally there is an output linear layer that helps determine the weights by providing the output for the given class.

In this work, the hyperparameters of the model are the embed dim, vocab size (determined by the tokenizer), hidden dimension for the LSTM, the output dim (number of LLMs), the number of layers for the LSTM, and the dropout rate used to prevent overfitting. Weight decay is also used in the optimizer routine. The following is a list of the hyperparameter values used for this work:

1. Embed dim = 128
2. Hidden dim = 256
3. Output dim = 7 (7 LLMs)
4. Number of LSTM Layers = 2
5. Dropout = 0.3 (30%)
6. Weight Decay = 1e-5

4.2 Optimization

The optimizer used was the Adam optimizer and the loss function was Cross Entropy Loss. Adam is typically considered a state of the art adaptive learning rate optimization function and cross entropy loss is typically used for multi-class classification problems such as this work. Varying learning rates were tested determine the one that achieved the highest performance. A learning rate of 0.01 was determined to be too high and a learning rate of 0.0001 was determined to be too low as both suffered poor performance in the validation sets. To get the most stable loss behavior, a learning rate of 0.001 was used for the optimization. The dropout weight and weight decay were also determined in a similar fashion. Part of the experimentation of this work was how the number of epochs for training affected the accuracy of the models on the different prompts. A total of 10 models were saved with the number of epochs varying from 5-50 (5, 10, 20, 30, 50 for few shot prompting and zero shot prompting). The validation accuracy for the model ranged from 50% to 94% depending on the number of epochs. 94% validation accuracy is quite high and this might indicate some potential overfitting which is to be discussed later. During training the loss of the model started around 2 and ended at about 0.01 at the highest number of epochs.

5 Experiments

The experiments in this work involve testing the effects of different types of prompting on the accuracy of the LSTM as well as how the number of epochs during optimization. The zero shot dataset are the prompts discussed in Section 3 just on their own. The few shot dataset are those same prompts but with an example as well. The accuracy of the classifier between the two datasets and the number of epochs is used as the metric for comparing the different models. In total there are 10 models to compare. Results and discussion is provided in the next section.

6 Results and Discussion

The full results are provided in Table 1. As can be seen in the table, an increase in the number of epochs does not necessarily mean an increase in the accuracy of the classifier. In the few shot dataset an increase in epochs does increase accuracy until beyond 30 epochs. This suggests that after 30 epochs there is some serious overfitting. The highest accuracy achieved by the few shot models is 52% at 30 epochs. This is also the highest accuracy of all the models that were tested. There is clearly some benefit in providing the models with example prompts when querying them. Notice also that there is significantly more variance between the accuracies of the one shot prompting versus the few shot prompting.

Epochs	Prompting	Accuracy (%)
5	few shot	8.69
10	few shot	33.83
20	few shot	41.94
30	few shot	52.00
50	few shot	24.57
5	one shot	42.06
10	one shot	45.00
20	one shot	48.69
30	one shot	45.60
50	one shot	49.37

Table 1: Experiment Results

It is suspected that this is caused by the fact that the few shot prompts may be directing the LLM to behave and finish prompts in a certain manner. This may cause less diversity in the dataset. Less diversity can make overfitting more of a problem and may be the reason that the variance of the accuracy is so high in the few shot dataset. However, providing examples may also cause major differences between the LLMs themselves so even though the dataset may be less diverse, the difference in the LLMs might be more distinct. Given an optimal number of epochs for training this may allow for higher classification accuracy.

As discussed earlier, the validation data accuracy on the model was roughly around 94% at the highest epochs. A high validation accuracy and low test accuracy for all the models suggests that each one was probably overfitting even with weight decay and dropout. This is probably because of the dataset being too small and the model finding trends too well. A larger dataset may help with the accuracy. Also, as discussed in some of the related work, classifiers that are fed texts with a higher number of maximum tokens also generally perform better. Since this dataset has prompts with max tokens between 30 and 100, it may not be capturing trends as well as it could if it had longer prompts.

7 Conclusions

This worked focused on building an LSTM classifier for a multiclass classification problem. This problem involved determining which LLM out of a subset of LLMs generated a given prompt. The dataset for this work was curated using a mix of APIs and open-source LLM models with a total dataset size of 14,000 points with each prompt having a random number of max tokens and temperature. The goal was to determine the accuracy of the model under different types of prompting (one-shot and few-shot) as well as how the number of epochs affected the model under these dataset conditions. Few-shot prompting provided the highest accuracy but was also more sensitive to the number of epochs needed for training.

Future work may include curating a larger dataset for this problem and comparing different hyperparameters among the LSTM model for classification accuracy. A dataset with longer prompt completions may also be beneficial so that the model can discover more variations between the different LLMs and their writing structures.

References

[1] Tiziano Fagni et al. "TweepFake: About detecting deepfake tweets". In: *PLOS ONE* 16.5 (May 2021). Ed. by Qingzhong Liu, e0251415. ISSN: 1932-6203. DOI: `10.1371/journal.pone.0251415`. URL: `http://dx.doi.org/10.1371/journal.pone.0251415`.

[2] Eric Mitchell et al. *DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature.* 2023. arXiv: `2301.11305 [cs.CL]`. URL: `https://arxiv.org/abs/2301.11305`.

[3] Irene Solaiman et al. *Release Strategies and the Social Impacts of Language Models.* 2019. arXiv: `1908.09203 [cs.CL]`. URL: `https://arxiv.org/abs/1908.09203`.

[4] Yi Tay et al. *Reverse Engineering Configurations of Neural Text Generation Models.* 2020. arXiv: `2004.06201 [cs.CL]`. URL: `https://arxiv.org/abs/2004.06201`.