CSE 584 Midterm

Maxwell Dailey

October 2024

1 Introduction

This document will describe how I acquired my data, how I built and trained my classifier, the results of my trained classifier, and related work that has already been done in this area.

2 Dataset Curation

To create my dataset, I started with the train.tsv file found at https://github.com/google-research-datasets/wiki-split/tree/master. This file contained one million English sentences extracted from Wikipedia edits (i.e. written by humans) in the format of two columns. The first column has a single sentence, and the second column has that same sentence split into two separate sentences. I chose to use eight separate Text Generation LLMs:
$openai-community/gpt2, utter-project/EuroLLM-1.7B, amd/AMD-Llama-135m, openai-community/openai-gpt, HuggingFaceTB/SmolLM-135M, Norod78/SmolLM-135M-FakyPedia-EngHeb, h2oai/h2o-danube3-500m-chat, Den4ikAI/DLM\_500m$
and wanted an overall dataset size of ten thousand, so I looped through the data in train.tsv until I had extracted 1250 unique sentences. For each of these sentences, I only kept the first five words, and these became my inputs for the LLMs I had chosen. I then created a DataFrame from the pandas library with two columns ("Text" and "Label"), where the "Text" column would store the output of the LLMs and "Label" would store the id I assigned to that LLM (e.g. 0,1,...,7). Thus, I ran each of the 1250 five-word sentence splits through each of the chosen LLMs, storing the outputted text and associated LLM label in the DataFrame.

3 Classifier and Training

The text in the DataFrame mentioned above is not usable as is because it must go from a string to a list of numbers. This is done with a tokenizer. I chose the "google-bert/bert-base-uncased" tokenizer on HuggingFace and used it to turn

all of my sentences into vectors of length 128 using padding.
Given the potential complexity of the task, I was unsure of the exact structure to use to maximize my classifier's performance on the data, though I knew that the input size should be 128 and the output size should be 8. Thus, I decided to break up my classifier into blocks of layers. Each block can be further divided into a learning sub-block and a down-sampling sub-block. The learning sub-block is defined as follows[1]:

$$y = ReLU(W^{(2)}Dropout(BN(ReLU(W^{(1)}Dropout(BN(x))))) + x)$$

And the down-sampling sub-block is defined as[1]:

$$z = ReLU(W^{(3)}Dropout(BN(y)))$$

Where $W^{(i)}$ is the weights of the fully connected ith layer and BN is a Batch Normalization layer. I also included a lone hybrid learning and downsizing sub-block at the end of the model with a set dropout rate of .2.
Given this general structural layout, using some validation dataset, I wanted to determine the optimal number of blocks, the optimal dropout value to use across all of the Dropout layers, and how many nodes, relative to the first fully connected layer in the block, should be in the second fully connected layer of the block. Before this could be done, though, I had to first split my data into training, validation, and testing and decide the scheduler and optimizer I wanted to use. For data splitting, after randomly mixing the rows of my DataFrame to ensure that each sub-dataset had a roughly equal number of each data category, I split the dataset with the following ratios: training = 72%, validation = 18%, and testing = 10%. I also decided upon a Cosine Annealing Learning Rate scheduler and Stochastic Gradient Descent as my optimizer. I now also wanted to determine the optimal initial learning rate, momentum value for my optimizer function, and batch size.
I determined all of these desired values using a technique called Bayesian Optimization, where I ran 300 trials spanning the ranges I gave for each of the variables I wished to optimize. For each trial, a set of values for each of these variables from each of their ranges is picked using an acquisition function, and a model with those characteristics is trained for some number of epochs (in this case, I set the number of epochs to a constant 100) on the training dataset. The model is then evaluated on its accuracy on the validation dataset to determine how optimal the chosen variable values are. At the end of all of the trials, the set of values that led to the best model performance on the validation dataset is returned as the optimal set.
With the optimal set of values found for each of the desired variables, I trained a new model using these values for 1000 epochs on the training dataset, saving the model every time it performed better than before on the validation dataset. Then, I evaluated the model's performance on the test dataset: overall cross-entropy loss, overall accuracy, accuracy for each category of data, false positive and true positive rate.

4 Results

The performance of my final trained model on the Test-Dataset is as follows:

Dataset	Overall Cross Entropy Loss	Overall Accuracy
Test Dataset	1.8730570127954707	0.3893229166666667

Table 1: Overall Model Performance Results

The accuracy of the model on data from each individual LLM is as follows:

Dataset	LLM 0	LLM 1	LLM 2	LLM 3	LLM 4	LLM 5	LLM 6	LLM 7
Test Dataset	0.3504	0.2807	0.2578	0.5000	0.1522	0.4000	0.2636	0.8505

Table 2: Model Accuracy By LLM

5 Further Analysis

To analyze the model's performance further on the individual categories, I created ROC Curves, which showed the true positive rate compared to the false positive rate for each category.

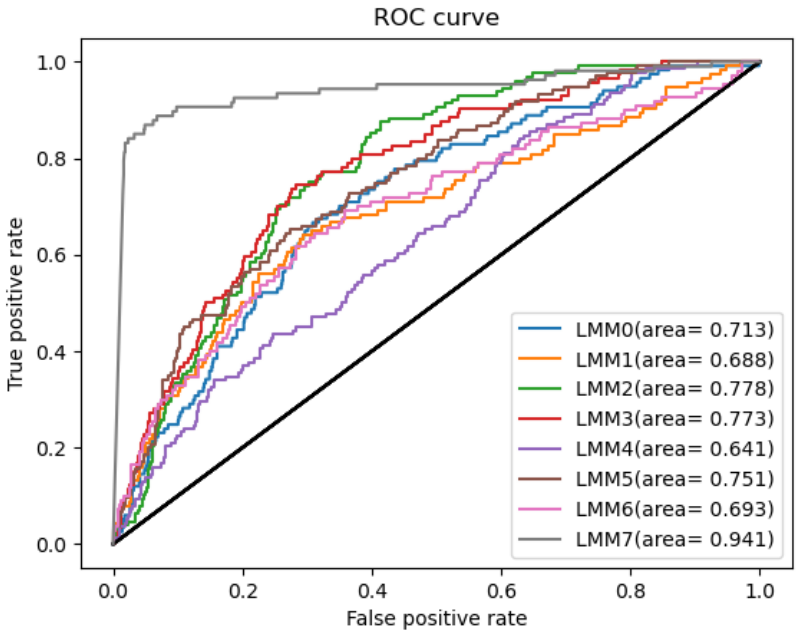

Figure 1: ROC Curves For Each Data Category

One interesting thing to note is that even though the model was less accurate on data from LLM 2 than data from every other LLM but LLM 4, it still had the second-highest AUC for LLM 2 data. This is not simply an outlier as other cases can be seen where the AUC for data from a certain LLM ranks relatively higher than its accuracy.

6 Related Work

One paper I found that relates directly to this topic is "Through the Looking Glass: Learning to Attribute Synthetic Text Generated by Language Models"[2]. In this paper, the authors set out to answer the question if similar approaches to the ones used to attribute texts to specific human authors could be used to attribute a generated text to its source LM. They proposed and tested several Machine Learning based attribution methods, with their best one being a fine-tuned version of XLNet, which achieved accuracy scores ranging from 91% to 98%. These accuracy results were achieved across a range of experiments using trained LMs, fine-tuned LMs, or varied text generation parameters.

While I struggled to find many papers on specifically using a classifier to differentiate between texts generated by different LLMs, I did find papers that analyzed the linguistic patterns and properties of text generated by LLMs. I believe these papers can apply to this topic as sentence completion does involve LLM text generation and the linguistic patterns and properties inherent to each LLM can be used (not necessarily in a DNN classifier) to determine which LLM a text came from at better than random chance. One paper I found that did this is "Contrasting Linguistic Patterns in Human and LLM-Generated News Text"[3]. In this paper, the authors generated text using six different LLMs across three families and four sizes. Their analysis spanned several measurable linguistic dimensions, such as morphological, syntactic, psychometric, and sociolinguistic aspects. Through these aspects, they were able to find measurable differences between human and LLM-generated texts, and were able to find measurable differences between the LLM models they were using. However, they do note that differences between LLMs and humans are larger than between different LLMs.

References

[1] Taewoon Kim. Generalizing mlps with dropouts, batch normalization, and skip connections, 2021.

[2] Shaoor Munir, Brishna Batool, Zubair Shafiq, Padmini Srinivasan, and Fareed Zaffar. Through the looking glass: Learning to attribute synthetic text generated by language models. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1811–1822, Online, April 2021. Association for Computational Linguistics.

[3] Alberto Muñoz-Ortiz, Carlos Gómez-Rodríguez, and David Vilares. Contrasting linguistic patterns in human and llm-generated news text. *Artificial Intelligence Review*, 57, 08 2024.