Detection of AI generated text

Vojtěch Slavík
ČVUT - FIT
slavivo4@fit.cvut.cz

December 19, 2023

1 Introduction

This project is on the public kaggle competition LLM - Detect AI Generated Text. In this competition the task is to determine whether an essay was written by a human or by an AI. The used model can be any chatbot.

2 Data

The dataset provided in the competition contains 10,000 essays, however only 1,000 are labeled and can be used fo training. According to the competition the essays contain human-generated ones and also LLM-generated ones (from various LLMs). However, only 7 essay prompts were used on all the data: "Car-free cities", "Does the electoral college work?", "Exploring Venus", "The Face on Mars", "Facial action coding system", "A Cowboy Who Rode the Waves", "Driverless cars".

Due to the low amount of essays in the training dataset it is crucial to use other data. Thankfully some kaggle users generated additional data - human and also LLM essays. These do not only contain the 7 prompts but also other ones. The data is available in the competition's discussion section and the ones used in this project are the following:

- Original dataset
- gpt-3.5 and gpt-4
- gpt-3.5 and student text
- llama and falcon

No data preparation was needed, except for renaming the columns, as the data was already in the correct format.

3 Methods

There are many approaches one can take to solve this problem. I decided to select the most promising ones and try them out. The first one is an encoder-based transformer model. The second one is an decoder-based transformer model and the last one uses TF-IDF and custom byte-pair encoding to vectorize the essays and then an ensemble of various models to classify them.

3.1 Encoder-based Transformer Model

This model comprises a pretrained backbone, the deberta-v3 model, and a custom head, which is a linear layer with a sigmoid activation function.

The model is trained on the training dataset using a custom training loop and evaluated on the validation dataset.

Due to constraints discussed in the results section, the validation data could not be used for parameter tuning. Instead, the test dataset was used, which is not the ideal approach as it can lead to overfitting. However, due to the nature of the competition, this was the only viable option.

The final hyperparameters are as follows:

- Learning rate (lr) for head: 5e-5
- lr for back: 3e-5
- Batch size: 32
- Epochs: 3
- Max length of tokens: 64
- Loss: Binary cross-entropy with logits
- Warm-up of lr: 1/10 of epoch
- No gradient accumulation

3.2 Decoder-based Transformer Model

This model also consists of a pretrained backbone, the mistral7b model, and a custom head, which is a linear layer with a sigmoid activation function.

The backbone was used in conjunction with quantization (a technique that reduces computation and memory cost of models) and low rank adaption (which also reduces the training and inference time). This was necessary due to the large size of the model (7 billion parameters) and the limited RAM provided by Kaggle (16 GB).

The model is trained on the training dataset using a custom training loop and evaluated on the validation dataset.

The final hyperparameters are as follows:

- lr for head: 5e-5
- lr for back: 5e-5
- Batch size: 16
- Epochs: 1
- Max length of tokens: 64
- Loss: Binary cross-entropy with logits
- Warm-up of lr: 1/10 of epoch
- No gradient accumulation

3.3 TF-IDF Model

This model uses byte-pair encoding to tokenize the essays. Uniquely, the tokenizer is trained on the test dataset. This strategy is employed to prevent overfitting on the training dataset and to enhance performance, as the test dataset contains data from all seven prompts.

The tokenizer is then used by TF-IDF to vectorize the essays. Finally, an ensemble of multinomial naive bayes, cat boost, SGD classifier, and LGBM classifier is used to classify the essays.

Unlike the first two methods, this approach was inspired by the work of other Kaggle users. However, it was included in this report for comparison with the other two methods.

3.4 TF-IDF

This model uses byte-pair encoding to tokenizer the essays. What is special here, is that the tokenizer is trained on the test dataset. This is done to prevent the model from overfitting on the training dataset and also to perform better (test dataset contains data from all 7 prompts).

The tokenizer is then used by TF-IDF to vectorize the essays. Finally, an ensemble of multinomial naive bayes, cat boost, SGD classifier and LGBM classifier is used to classify the essays.

In contrast to the first two methods, this approach was not thought out by me but was inspired by the work of other kaggle users. However, I wanted to include this approach so that I can compare it to the other two.

4 Results

In this competition the metric used is the area under the ROC curve (AUC). During the training and testing of the models I observed an interesting phenomenon.

The AUC on the validation dataset easily reaached 0.998+ but the AUC on the test was much lower (0.7-0.8). This was the case for the encoder and decoder models.

However, the TF-IDF model performed much better on the test dataset. This led me to believe that the test dataset contains data specifically designed to fool the LLMs. I tried to remedy this by using external datasets with different prompts, data generate by different LLMs and also by using LanguageTool library to correct the essays. However, this did not help and the AUC on the test dataset remained low.

In the end these are the results I got:

Model	Val	Test	Train [mins]	Infer [mins]
Enc-based	0.998	0.754	20.0	3.3
Dec-based	0.999	0.799	47.5	42.5
TF-IDF	0.999	0.961	6.95	193.4

Table 1: Results

5 Conclusions

The encoder and decoder models performed very well on the validation dataset but horribly on the test dataset. This is probably due to the fact that the test dataset contains data specifically designed to fool the LLMs and also due to the fact that the byte-pair encoding tokenizer was trained on the test dataset, which led to better generalization.

The TF-IDF model also had the fastest training time, but the slowest inference time. This is because the TF-IDF model is an ensemble of various multiple models.

When comparing the encoder and decoder architectures we can see that the decoder model is way slower but also performs better.

However, these conclusion cannot be applied to generalized AI text detection. This is due to the fact that the test dataset contains data specifically designed to fool the LLMs and also the fact we have an access to the prompts used for the generation. So it cannot be said that TF-IDF is generally better than transformer models for AI text detection.

But at the same time it gives us some insight into the problem and shows us that even the most advanced transformer models can be fooled by specifically designed data and that we do not always need

transformers for NLP tasks.

What I would like to try in the future is to use bigger variety of data to test these models, which might give a better insight into the problem and offer more robust solutions for general AI text detection.