Luis Kolb <01622731> 18. Jan. 2023

https://github.com/LuisKolb/adl-ws-2022

distinguishing **human-authored** text from **machine-generated** text

Applied Deep Learning WS2022

Introduction

Computer-generated content is becoming an ever more relevant concern as the generating language models continue to improve - being able to distinguish if a piece of content originally came from a human or a machine is important. Not only for basic ethical concerns regarding journalistic integrity, but to increase acceptance by the general public as well. Trust in these new systems can only be achieved by increasing transparency, and being able to identify such a system “in the wild” is a step towards this goal.

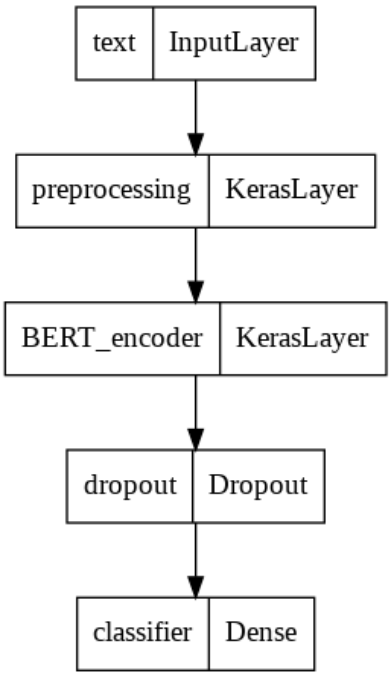
Unfortunately, humans themselves are rather bad at this task, as Crothers et al. state in Machine Generated Text: A Comprehensive Survey of Threat Models and Detection Methods. Though deep learning, intuitively, seems suited to attempt to solve this problem. Another comparison of different detection approaches in various settings was published in a very recent paper by Uchendu et al. here: Attribution and Obfuscation of Neural Text Authorship: A Data Mining Perspective. They call this type of problem "Authorship Attribution".

In addition, rather than a diverse range of datasets and models like the ones provided at https://turingbench.ist.psu.edu/, I wanted to purely focus on detecting GPT-3 generated text. The reason for this decision is the relative recency, impressive performance and popularity in media the model has enjoyed. Additionally, outputs of older models like GPT-2 are - at least compared to newer models - easy to detect.

As a side note, at the time I started this project, Chat-GPT was not yet released. Even more than its predecessor models, there was extraordinarily widespread media coverage, especially in connection with cheating in classrooms. This is a prime example of the “authorship attribution” problem, and highly relevant **today**. Keeping up with the rapid improvements in this space is proving an enormous challenge, as some schools found no better solution than simply blocking the OpenAI website on their school network.

The dataset to train the deep learning model was created “from scratch”, in line with the *bring your own data* approach. As opposed to, for example, news stories (a popular "type" of text), I want to build a dataset of **tweet replies**. The setup is:

- use the Twitter API to sample a (usually) human-authored “reply tweets” to “parent tweets”
- and then prompt GPT-3 to author a reply to the same “parent tweet”



With this setup, we can generate text passages that are created “equally” in the same context. The datasets generated through this method could be used in more sophisticated models comparing the human vs. machine passages side-by-side. However, for this project, I just implemented a basic BERT encoder and classification head setup (see →), since the aim of this project is to provide the tools to more easily generate the data yourself.

Since I'm still unsure if tweet text can be shared in this manner according to the Twitter TOS, instead the project has a function to make a file containing

- the original tweet ID
- the human reply tweet ID (with text stripped out)
- and the GPT-3 generated text reply to the tweet

from the output files left. Such a data set is available attached to a release on GitHub as a .tsv file. The tweet text for the human reply will need to be fetched separately using the IDs if required. For most Twitter data sets online, just providing IDs is the usual approach.

More information on how to use the scripts in the project is provided in the README of the GitHub repository.

Creating the dataset

The main part of this project was building the tools to create a dataset using your own Twitter API and OpenAI API keys. Starting out on OpenAI, I got $18 worth of “credits”, and Twitter API credits were not a problem at this scale. Being limited by OpenAI credits, the biggest dataset I created consists of **6344** completions from GPT-3 using the **text-davinci-003** model (the most popular and complex one - at least before ChatGPT released to the public). The dataset additionally contains again as many “human-authored” tweet replies, and the records are labeled **human** or **machine**. As stated above, they are released in an incomplete format due to Twitter TOS.

For my deep learning experiments, I used the complete dataset, which my scripts by default create in a structure that can easily be loaded into a data set using the tensorflow function “*tf.keras.utils.text_dataset_from_directory()*”.

As we will see in the next section, some preprocessing is necessary, which is not by default done by the dataset creation scripts in this project.

Some additional preprocessing could be implemented (or required), like removing records that are too short to be useful, more experimentation would be needed to know.
An additional concern is the lack of variation in the data, since GPT3 completions were alway generated with the same “template” prompt:

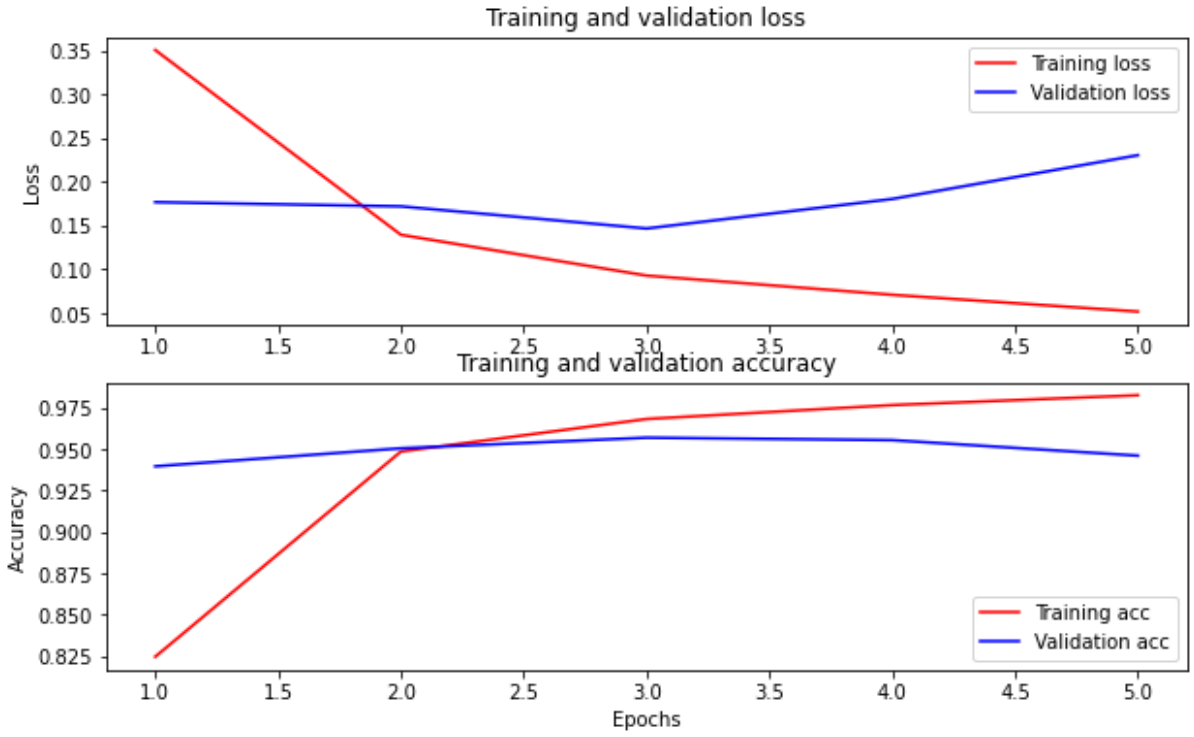
```
Write a reply to this tweet in less than 280 characters: "{tweet}"
Reply:
<the GPT3 completion here>
```

This could have introduced some bias into the completions, and maybe this point would need to be addressed. Completion requests to the model always used a temperature of **0.9**, to generate diverse answers.

Running a model

I ran two experiments with the dataset where I implemented just a very basic pre-trained small BERT transformer encoding layer, and fine-tuned it on my dataset. Then, for binary classification, we add a dropout and a dense layer as the output layer. This is a very simple setup, yet surprisingly effective.

This is the progression of loss and accuracy over the training epochs in the 2nd experiment:



In my two experiments, the best results for binary classification accuracy were:

- **0.9747:** this was on the first "test" experiment, where no preprocessing was done - could probably be achieved just by looking at the first few letters of the reply, since most of the machine-generated text didn't start with @user_i_am_replying_to, while most human-generated replies did.

- **0.9457:** this was for the "actual" dataset with ~6000 records each for human and machine generated tweet replies, with the default 80-20 train-test split. This dataset also had any @usernames stripped, to remove the advantage introduced by the Twitter API mentioned above.

Surprisingly, even with the @usernames stripped, the classification accuracy is still really good. Seeing as more sophisticated models achieve comparatively small accuracy scores of less than 0.8 (on turingbench), this was odd to me, as I had expected no better than 0.7 binary classification accuracy. It also indicated that the data may be flawed in the way it was generated, see the point about the completion prompt above. Another possibility is that the tweet replies are too short, or that GPT3 “writes” rather structurally predictable answers, while Twitter replies are more… “diverse” in their structure, language used and emoji utilization.

A possible setup to further validate the quality of the data could be to use the full context of
- “parent tweet” and
- “human reply” vs. “machine reply”

to let a network decide which reply is human-authored and which is machine-generated, or which is a better “match” to the parent tweet.

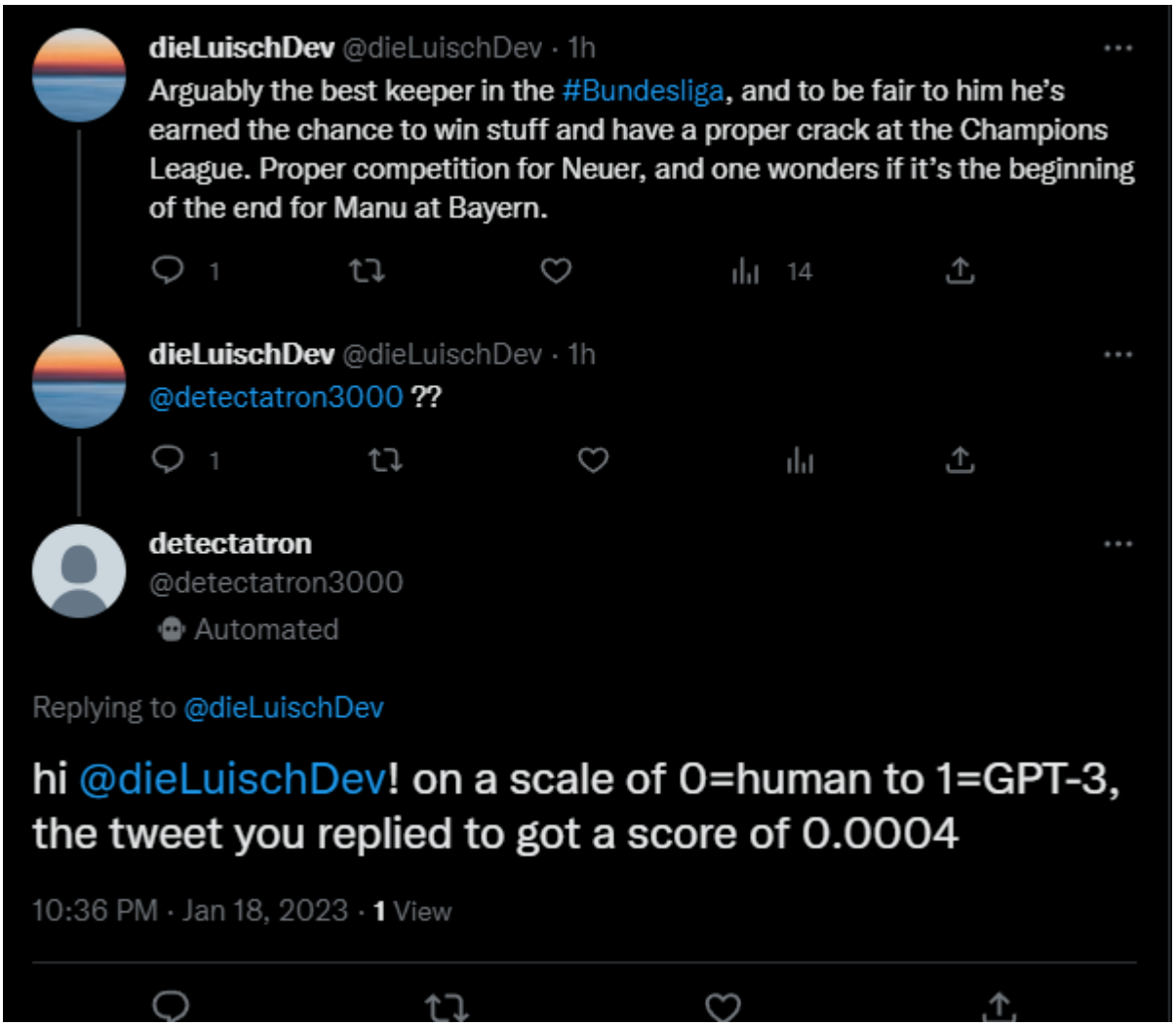
@Detectatron3000

To demonstrate the basic model I had obtained, I created a bot on Twitter, which seems like a fitting way to demonstrate how tools like this could be used and accessed.

Tagging/Mentioning it “under” a tweet makes the bot reply to your tweet with the score the “above” tweet got on a scale of 0 (human) to 1 (GPT3).

Disclaimer: the bot only replies when I manually run the script to make it check and reply to any new mentions.

Here it is casting its judgment once invoked:



Conclusion

If I would do the same project again, I would change a few things (especially looking at possible future development):

- Before starting the implementation, I'd look at how popular frameworks expect data to be structured, and choose an output format which conforms to one of those. This would make it easier to publish the data.
- Look into dataset repositories where I could make the data publicly accessible on a more purpose-built platform, as opposed to a simple GitHub release.
- Provide more options to customize the way completions are generated, which would maybe enable me to easily run automated experiments to evaluate the impact of more varied completion prompts on a deep learning pipeline.
- Continuing from the point above, I'd focus more on building a more automable pipeline from API calls to model evaluation to more easily test different dataset creation setups, which could also help with evaluating detection on different language models and APIs.

Time breakdown

For this project, the time breakdown after finishing the 2nd assignment looked like this:

- **building methods to pull tweets and gpt-completions:** ~3 days, with some additional effort of ~1 day to work out bugs
- **implementing a DL pipeline:** ~1.5 days, most of the time spent on finding a way to make the data set format compatible with the tensorflow data loader functions
- **finetuning:** ~0.5 days, due to sickness this part unfortunately had to be cut short, only stripping @usernames was implemented as a preprocessing step, and no model customization was possible
- **documentation:** ~1 day for cleanup, code documentations

After completing this 3rd, final assignment, I spent

- 2 days on implementing the **twitter bot**
- 1 day writing this **report** and recording the **presentation:** https://youtu.be/sAPBFAO1leY

I definitely underestimated how complicated it would be to load my data into a format compatible with tensorflow or pytorch, and ultimately decided to use tensorflow - simply because I found it relatively easier to create a compatible data set using the provided interfaces in that library. Even so, I had to rewrite a major part of my output format and structure, since the most convenient (to me) interface relies heavily on semantic directory structure (a single record is saved to a single file, class names are based off of directory names, etc.) whereas I previously used just a single (or a few) .tsv file(s) to output data to.

Overall, the only unexpectedly large deviation from the expected work plan in assignment 1 was in tuning, as I expected to spend a lot more time building and training a model. Using the pretrained BERT transformer encoding layer cut out a lot of training time and was very convenient to implement and get a basic model to spit out predictions for the demo twitter bot.