GPT Generated Text Detection

Nan Li (22-736-169)
Yiqin Zhang (22-738-397)

May 5, 2024

Abstract

In an era where AI-driven language models like GPT have become increasingly prevalent, the need for effective GPT detection techniques has never been more critical. In this project, we collected textual data generated by both GPT-3 and GPT-3.5. We used different language models to vectorize texts and trained several models to classify if the text is human-written or GPT-generated. An interesting discovery is that, contrary to our expectations, GPT-3.5 did not perform more like a human than GPT-3. In other words, our best-performing model achieved higher accuracy in distinguishing GPT-3.5 generated text from GPT-3 generated text compared to human-written text.

Figure 1: Interpretation of Two Female Data Scientists Trying to Defeat ChatGPT

1 Introduction

This project detects whether a paragraph is written by a human or generated by GPT. The original dataset consists of articles from Wikipedia [1] and GPT-3 generated text using the same articles' starting sentence (known as prompts). We created a second dataset from extracting texts from GPT-3.5 via OpenAI's API using the same prompts. We processed, trained, and evaluated the datasets using 4 vectorization techniques and 13 classification models and finally compared and contrasted the performances between models and datasets.

In this report, we summarize the experiment process and key findings. Details regarding implementation can be found in our project repository: GPT_Generated_Text_Detection [1].

2 Data Set

We chose the *GPT-wiki-intro* dataset [1] from Huggingface. The dataset was constructed according to the Wikipedia dataset [2] and GPT-generated text based on that. The dataset contains Wikipedia introductions, which we use as human-written, and GPT (Curie) generated introductions. We know that Curie is founded upon GPT-3 and does not achieve the same level of excellence as GPT-3.5. So we create our own dataset using OpenAI API and GPT-3.5-Turbo and the same prompts from the original dataset (codes can be found in *chatgpt_api.ipynb*, prompts used for generation are in *GPTprompt.pkl* and responses are in *GPTresponse.pkl*).

The original *GPT-wiki-intro* dataset contains some columns we don't need, so we only selected 'wiki_intro' (Introduction paragraph from Wikipedia) and 'generated_intro' (Introduction generated by GPT (Curie) model). Due to computer limitations, we have chosen to work with a subset of the dataset, specifically 1500 rows, dividing them into 1500 positive and 1500 negative data points. Additionally, the prompts column was used to generate text from GPT-3.5 and also generate a dataset that contains 1500 positive and 1500 negative data points.

To introduce randomness into the datasets, we shuffle the 3000 data points for both 2 datasets to break the original order of the dataset. And we take the first 300 data points out from each dataset for hyperparameter search since using all the data will take a lot of time (codes can be found in *data_prepare.ipynb*).

3 Preprocessing

We did normalization to all text data, which includes removing extra space, making all letters lowercase, and removing emojis. These normalization steps were taken in the data preparation stage. And then we use 4 methods to vectorize the normalized texts.

1. **Bag of Words:** In the BoW model, a text document is treated as a collection of words, and the order of words is ignored. The main idea behind BoW is to represent a document as a histogram of word occurrences. scikit-learn [3] offers a utility known as "sklearn.feature_extraction.text.CountVectorizer", which we employ in our project. The CountVectorizer first tokenizes the text data, then builds a vocabulary, and last counts word occurrences. However, the vocabulary size for all words is 36862 for GPT-3 dataset, which is too long for the model to train. So we ignore words that have a document frequency strictly lower than 15. Then we have the vector length as 2815 for GPT-3 dataset and 4199 for GPT-3.5 dataset (code can be found in *S2V_BoW.ipynb*).

[1] Github Repository: https://github.com/yvonne-yiqin-zhang/GPT_Generated_Text_Detection

2. **Term Frequency-Inverse Document Frequency:** TF-IDF is an enhancement of the BoW model that takes into account the importance of words in a document relative to their frequency in a corpus. It is particularly useful for identifying the significance of words in a document within a larger collection of documents. We also use the scikit-learn function "sklearn.feature_extraction.text.TF-IDFVectorizer". The same length issue of BoW happens in TF-IDF too, so we used the same limit for this model (code can be found in *S2V_TFIDF.ipynb*).

3. **Sentence-BERT:** Unlike traditional methods such as BoW or TF-IDF, SBERT [4] aims to capture semantic similarity and context in sentence representations. It's an extension of the popular BERT (Bidirectional Encoder Representations from Transformers) model, which was originally designed for word-level embeddings. But SBERT has been adapted and can be used with text embedding tasks (code can be found in *S2V_SBERT.ipynb*).

4. **Sent2Vec:** Sent2Vec[5] is another Python package that provides a simple interface for sentence embeddings. We use the pre-trained weights 'distilbert-base-uncased' in this project which is also a BERT model. But the weights only support text no longer than 512 chars, so we only take the first 512 chars in every text (code can be found in *S2V_sent2vec.ipynb*).

We used the 4 methods to vectorize all the 3000 data points from both datasets and save them for the model training.

4 Model Training

Leveraging the extensive model inventory available in scikit-learn [3], we experimented with thirteen algorithms for this classification problem. Broadly speaking, these algorithms can be grouped into four categories.

1. **Linear Models:** If the underlying relationship between these vectors is relatively simple, linear models can be a good starting point to uncover these relationships. *Logistic Regression* (LR) applies logistic sigmoid function to transform output to probability value which then is mapped to binary classes. *Ridge Classifier* applies L2 regularization to linear regression to prevent model overfitting. With single-layer and a linear activation function, *Perceptron* can be seen as a linear model with linear decision boundary. *Stochastic Gradient Descent (SGD)* implements regularized linear models with SGD learning.

2. **Decision Tree and Ensemble methods:** Tree-based methods often perform very well in classification tasks. *Decision Tree* mimics the structure of a tree to make decisions based on input features which also has the advantage of explaining the decision process. For more complex problems, ensemble methods such as *Bagging*, *Random Forest*, *Adaptive Boosting Classifier* (AdaBoost) and *Gradient Boosting* (GBoost) can improve model accuracy though differ in the methods used. For example, *Bagging* uses bootstrap resampling to ensemble different decision trees independently and voting to come up with the final prediction whereas *Random Forest* is an extension of bagging but also introducing randomness by only considering a subset of features during each decision split.

3. **Neural Network:** *Multi-layer Perceptron* (MLP) builds on top of *Perceptron* and allows multiple number and size of layers, as well as allowing flexible options of activation functions such as 'logistic', 'tanh' and 'relu'. This model should be able to capture more complex relationships between input data.

4. **Miscellaneous:** *K Nearest Neighbours* (KNN) can be an easy but effective algorithm to detect relationships between input features. With non-linear kernel, *Support Vector Machine* (SVM) is another powerful algorithm which has non-linear decision boundary and can learn complex relationships. *Gaussian Naive Bayes* (GaussianNB), on the other hand, is a probabilistic algorithm that is particularly well-suited for data with continuous features when the Gaussian distribution of each feature is not severely violated.

We first implemented a function **compute_GS** to perform hyperparameter search for each vector dataset and each model mentioned above. We utilized *GridSearchCV* with *Cross Validation* (CV) from scikit-learn [3] to select optimal model parameters with the highest *f1* score. The function returns the list of models initialized with best-performing parameters. For the dataset built based on GPT-3 alone, we performed 156 hyperparameter search (4 vectorized dataset x 13 model x 3 cross-validation splitting strategy). We further analyze these results in more details below. We use the smaller dataset (300 samples) to perform parameter search. After obtaining the best parameters, we use the larger dataset (3000 samples) to perform model training.

We perform the same parameter searching and model training procedures on dataset extracted from GPT-3.5 in hope of comparing and contrasting model performance. Codes can be found in *train_model.ipynb* in our project repository.

5 Evaluation

We evaluated our classification models based on four evaluation metrics: *accuracy*, *precision*, *recall* and *f1_score*.

1. *Accuracy* measures the overall correctness of the model's prediction. It can be a good metric when the dataset is balanced which is our case here with the same number of human-generated and bot-generated text data.

2. *Precision* quantifies the accuracy of positive predictions made by the model. It can be a good metric when you want to steer the model to have fewer false positives.

3. *Recall* quantifies the model's ability to identify all relevant instances in the data. It can be a good metric when you want the model to capture as many of the actual positives as possible.

4. *F1 score* is the harmonic mean of precision and recall. This can be a good metric in general as this single score balances precision and recall.

We select *F1 score* as our main evaluation metric since it is a more balanced score between precision and recall and during our analysis, we did not observe a significant variation between accuracy and f1 scores (i.e. model performs well in f1 score also performs well in accuracy and vice versa). For more details, refer to *result_analysis.ipynb* in our project repository. We further rely on test F1 score only since it is less biased than the train score.

We implemented a function **compute_model_performance** that takes a list of models initialized with fine-tuned hyperparameters, training data and labels, performs *cross_validate* from scikit-learn [3] and saves average validation scores in a log. For the dataset built based on GPT-3 alone, we performed 156 model train and evaluation each of which uses the model configuration returned from **compute_GS** (4 vectorized dataset x 13 model x 3 cross-validation splitting strategy). We further analyze these results in more details below.

For evaluation, we also perform the same evaluation procedures on dataset extracted from GPT-3.5 in hope of comparing and contrasting model performance. Codes can be found in *train_model.ipynb* in our project repository.

6 Analysis and Conclusion

Upon reviewing all the experiment results, we summarized our findings hereunder. Majority of the findings are based on results using GPT-3 dataset but generally applicable to GPT-3.5 as well. We discuss some GPT-3.5 specific findings in point 6.

1. Model performance does not vary significantly between vectorization methods. We thought *TF-IDF* should perform better than *BoW* since *TF-IDF* has more information on frequency. But **BoW** performs better on several models. We also would think *SBERT* and *Sent2Vec* would be better since they use the popular pre-trained model *Bert*, but they seem to work worse with most models.

2. No one best performing model across four vectorization methods, but overall **GBoost** seems to perform the best for two out of four vectorized datasets. For *BoW* and *TF-IDF*, GBoost performs the best whereas Ridge and SGD performs the best for *SBERT* and for *Sent2Vec* respectively. We see **KNN** and **Ridge** perform considerably worse in two out of four vectorized datasets respectively with KNN having the worst f1 score. Their simplicity may be the contributing factor for their poor performance.

3. GBoost is super time consuming to train. It is not surprising that ensemble methods takes longer to train due to its complexity. However, we see as much as 17x longer training time than other ensemble method such as *Random Forest*. One should be cautious about the trade-off between slightly higher accuracy v.s. long training time.

4. We also note SVM.SVC has significantly higher scoring time than other models (37x higher than average excluding SVC). This may be explained by the use of non-linear kernel (i.e. 'rbf') and the number of support vectors which demand higher computation complexity.

5. We observe overfitting across all models and increasing cross validation from 3 to 8 helps reduce overfitting in some instances. Taking train/test as a benchmark for overfitting ratio, we examine model overfit across all models, all vectorized datasets and all validation strategy. We note Ridge overfits *BoW* and *TF-IDF* consistently and higher the number of CV, higher the overfitting ratio. For *SBERT* and *Sent2Vec*, Bagging overfits the most across all combination except *SBERT* with cv=3 and cv=5 where KNN demonstrates higher ratio. Increase our dataset size should effectively reduce overfitting, however, would also drastically increase computation time.

6. Contrary to our expectation, the models performed much better on datasets extracted from GPT-3.5. As can be compared in Figure 2, the best model achieved 99% f1 test score with *BoW* vectorization technique. Furthermore, **Ridge** consistently perform the best on *SBERT* whereas **LR** performs the best on *BoW* and *TF-IDF*, and **MLP** performs the best on *Sent2Vec* vectorized dataset. This suggests that even though GPT-3.5's performance is impressive to a human, it has very strong characteristics (e.g. words, sentence patterns and etc) that can be easily deciphered by machine learning models, sometimes as simple as a Logistic Regression.

Detailed model results on GPT-3 datasets can be found in 2a and GPT-3.5 in 2b. In our project repository, *result_analysis.ipynb* contains codes and *experiment_result* folder contains all raw result logs.

Based on the cross validation results above and for each vector method, we selected the best performing model with optimal hyperparameters as our 'chosen' model (highlighted in yellow in Figure 2, separated by GPT-3 and GPT-3.5). We randomly selected 50 prompts and a total of 100 sample data (1 generated and 1 wiki text) from our dataset as the test set and used the remaining as the train set (i.e. 2900 data samples). The

test classification results are summarized in Figure 3. It is obvious to see our model performs better on detecting GPT-3.5 generated texts with higher accuracy and F1 score across the board. This is consistent with our cross validation results but remains opposite from our initial expectation. *Turbo_tfidf* and *Turbo_bow* achieve (almost) perfect classification results which may suggest the similar underlying methodology used in these two vectorization methods.

In attempt to understand the difference in performance between GPT-3 and GPT-3.5, we filtered out the false negatives from test set *bow* and compared GPT-3, GPT-3.5 and Wiki texts. False negatives are the samples that are generated by GPT but falsely classified by our model as written by human. Not surprisingly to our belief, neither GPT-3 nor GPT-3.5 generated text is distinguishable to human eyes. They all read fluently, coherent and have no grammatical errors which are all strong arguments to support this detection being a difficult task. However, the hidden but strong characteristics within these texts are astonishingly easy for machine learning models to pick up. *test_dataset_deepdive.ipynb* in our repository shows discrepancy results and displays one text example for comparison.

	fit_time	score_time	bow	sbert	sent2vec	tfidf
LogisticRegression	0.740724	0.012210	0.843510	0.832632	0.837207	0.815058
RidgeClassifier	0.718735	0.013671	0.638305	0.836479	0.836138	0.606822
SVC	7.973025	1.212476	0.831069	0.794456	0.814187	0.828944
SGDClassifier	0.377191	0.008848	0.712951	0.827102	0.855611	0.742623
Perceptron	0.263325	0.008625	0.826095	0.805412	0.822953	0.803903
GaussianNB	0.127342	0.024045	0.704109	0.762862	0.666572	0.655411
DecisionTreeClassifier	1.419719	0.010129	0.723478	0.677456	0.618571	0.704376
BaggingClassifier	16.766576	0.051518	0.756759	0.673506	0.681701	0.762643
AdaBoostClassifier	13.695222	0.067691	0.776707	0.736396	0.711634	0.816977
RandomForestClassifier	3.599806	0.031297	0.809064	0.741940	0.733237	0.823209
GradientBoostingClassifier	61.600051	0.011388	0.859328	0.755817	0.762536	0.830987
KNeighborsClassifier	0.049500	0.142565	0.667259	0.465817	0.532127	0.633688
MLPClassifier	5.754603	0.015385	0.844577	0.806582	0.843068	0.826161

(a) Model Result GPT-3

	fit_time	score_time	Turbo_bow	Turbo_sbert	Turbo_sent2vec	Turbo_tfidf
LogisticRegression	0.650146	0.012287	0.989559	0.904674	0.928932	0.984913
RidgeClassifier	0.898047	0.012986	0.958488	0.924630	0.925536	0.948445
SVC	5.812186	0.630767	0.987873	0.894934	0.918265	0.984642
SGDClassifier	0.344514	0.007789	0.945249	0.904840	0.929989	0.946857
Perceptron	0.333724	0.009340	0.968626	0.897830	0.920320	0.968223
GaussianNB	0.196417	0.026213	0.920379	0.812482	0.740143	0.909343
DecisionTreeClassifier	2.305436	0.009308	0.925407	0.620811	0.726481	0.907942
BaggingClassifier	22.789661	0.078301	0.938891	0.655749	0.797464	0.947101
AdaBoostClassifier	14.765375	0.105328	0.978304	0.724023	0.812904	0.978179
RandomForestClassifier	3.444786	0.028017	0.982013	0.788492	0.816458	0.982278
GradientBoostingClassifier	73.220426	0.011399	0.977998	0.807658	0.855746	0.981566
KNeighborsClassifier	0.069485	0.167508	0.495802	0.683325	0.728536	0.178940
MLPClassifier	5.984691	0.015926	0.987216	0.908223	0.930150	0.976146

(b) Model Result GPT-3.5

Figure 2: Summary of F1 test scores with 8-fold cross validation from (a)GPT-3 and (b)GPT-3.5 datasets. *fit_time* and *score_time* compute the averages of four vectorized datasets. Highest performing models are highlighted in yellow for *BoW*, *SBERT*, *Sent2Vec* and *TF-IDF* respectively.

	sbert	bow	sent2vec	tfidf	Turbo_sbert	Turbo_bow	Turbo_sent2vec	Turbo_tfidf
accuracy	0.840000	0.92	0.840000	0.92	0.930000	1.0	0.890000	0.990000
f1	0.846154	0.92	0.846154	0.92	0.929293	1.0	0.884211	0.990099
recall	0.814815	0.92	0.814815	0.92	0.938776	1.0	0.933333	0.980392
precision	0.880000	0.92	0.880000	0.92	0.920000	1.0	0.840000	1.000000

Figure 3: Test result for each vectorized datasets. Columns with 'Turbo' indicate GPT-3.5 dataset.

7 Future work

Comparison between datasets extracted from GPT-3, GPT-3.5 and GPT-4 can yield interesting results. This analysis can also be extended to other language models such as BingAI and LaMDA. Further deep dive into the reasons behind better model performance on GPT-3.5 dataset would be highly valuable because it requires deeper understandings of the inner workings of these fantastic language models. One such method could be to utilize model-agnostic explainable algorithms such as LIME or SHAP to determine importance components of a sentence. The total time spent on parameter search and model training took just over seven hours with a Duel-core Macbook using a relatively small dataset. With higher computation capability, a much larger dataset can be utilized during model training and evaluation which should reduce model overfitting and further increase accuracy.

References

[1] Aaditya Bhat, "Gpt-wiki-intro (revision 0e458f5)," 2023. [Online]. Available: https://huggingface.co/datasets/aadityaubhat/GPT-wiki-intro

[2] W. Foundation. Wikimedia downloads. [Online]. Available: https://dumps.wikimedia.org

[3] G. Varoquaux, L. Buitinck, G. Louppe, O. Grisel, F. Pedregosa, and A. Mueller, "Scikit-learn," *GetMobile: Mobile Computing and Communications*, vol. 19, no. 1, p. 29–33, Jun 2015.

[4] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[5] P. A. Alberto Marengo, "sent2vec: Unsupervised Learning of Sentence Embeddings," https://pypi.org/project/sent2vec/, 2022, pyPI Python Package.