Ritika Kumar ⁕ Xin Wang ⁕ Ardavan Mehdizadeh

Who Wrote This?

A Human vs AI text classifier ; ***GitHub:*** [Link]

Summary

With the rise of large language models and generative AI, there is a significant increase in the amount of text data generated by AI. In a world where literature is considered an art that generates monetary value for multiple brilliant writers across the world, AI generated text often devalues the authenticity of literature. The text data generated by AI is usually difficult to identify due to it being highly eloquent and grammatically correct. However, AI generated text lacks the depth and originality, and misses upon the nuances of human touch often found in authentic written content.

WhoWroteThis is a python package that allows a user to distinguish text data as Human or AI generated. It gives the user an easy to use method to ensure authenticity and maintain the integrity of literature and written content.

Methodology and Design

The idea behind the python module is the designing of a AI vs Human text classifier trained on a labeled dataset of human and AI generated text data. The pipeline for the project consist of three major executables:

1. **Text embeddings**

   We used the two datasets, 'AI vs Human Text' dataset that consists of 487,235 rows and 'DAIGT V2 Train Dataset' that consist of 44,868 rows with two columns indicating text and the corresponding label (0-Human, 1-AI).

   The text data was first padded or truncated to 1024 tokens, then converted into tokenized inputs using the Tokenizer of GPT2. The tokenized inputs are then converted to text embeddings using GPT-2 where each text embedding has a shape of (#tokens-1024, 768) corresponding to each row of text data. The pooled embeddings are then obtained using mean pooling to get an embedding with a shape of (1, 768) for each text.

2. **Models**

   We used the GPT-2 text embeddings to train a classification model to predict the label of the text data. The merged dataset obtained has 305,796 data points labeled as human generated, and 198,935 data points labeled as AI generated. To tackle the issue of imbalanced classes, we used the ensemble method of boosting to iteratively train a strong model by sequentially training weak learners. We implemented this using the following boosting models:

   - XGBoost: We used XGBoost, a popular library for implementing gradient boosting algorithms.
   - LightGBM: We used LightGBM for increasing the speed and efficiency of training the data. LightGBM grows trees leaf-wise instead of level-wise, which can lead to faster convergence and reduced memory usage. This is beneficial for datasets with a large number of features such as ours.
   - AdaBoost: We used AdaBoost or Adaptive boosting model on merged dataset. AdaBoost is effective because it focuses on difficult-to-classify examples, effectively adjusting its learning strategy based on the mistakes made by previous classifiers.

3. **Model evaluation (Compare multiple models on unseen data)**

   We obtained unseen texts by merging multiple datasets (LLM: 7 prompt training dataset, LLM: Mistral-7B Instruct texts) and removing the duplicates in these new datasets which might have already been seen by the models during training. We sampled 10000 texts (5000 AI texts, 5000 Human texts) out of the unseen texts. Based on these 10,000 samples, we tested our models on both raw text embeddings and preprocessed text embeddings.

Usage

The package was created with the aim of flexibility of the user to utilize it for multiple purposes. The package consists of a TextPreprocessing module that can pre-process text data for users and convert it into tokens.

The processed data can then be converted into embeddings using the TextEmbedding module. The default model for extracting

```
def main():
    processor = TextPreprocessing("test.txt", file_given=True)
    print(processor.preprocess())
    text = "This is another text to be processed!\n Hope this also works well."
    processor.set_text(text)
    print(processor.preprocess())
    processor.set_text("test.txt", file_given=True)
    print(processor.preprocess())
```

embeddings is GPT-2 but the module supports other text embedding models including InstructorXL, longformer-base-4096 and bert-based-uncased and can be changed as per the user's initialisation of the TextEmbedding class.

The users can use the embeddings obtained from the TextEmbedding module or their own embeddings and use the Classifier module to predict whether the text data is Human or AI generated.
We are using all our trained models to create a list of predictions. These models then undergo a voting system where the prediction in majority wins. The number of models for prediction are odd numbered to ensure a tie-breaker during each prediction cycle.

```
def main():
    embeddings = TextEmbedding( text_file: 'text.txt', model='gpt-2').get_embeddings()
    predict = Classifier(embeddings)
    print(predict.predict_text())
```

Discussion

There are a number of tools that pre-exist for AI text detection. However the main focus of our project was to create a versatile library to ensure flexibility for users to not just predict a text class but enable them to choose their own LLM model, generate text embeddings and evaluate the models after prediction. WhoWroteThis provides a segregation of tasks involved in text classification that enables multiple functionalities including pre-processing text data for NLP tasks, generating text embeddings from custom data and evaluating model accuracy on test datasets.

Apart from being useful in creating a workflow for text classification tasks, the library can also be used as a unified unit for a user-friendly way of AI text detection. We used streamlit to create a user interface that enables the user to open a web application on their local server. This web app allows the user to enter text data and display predictions. It also enables the user to see the visualizations of model accuracy and the comparison of all the models.

Statement of contributions:

Xin Wang:

- Wrote the codes for getting tokenized inputs and text embeddings using GPT-2. Wrote the codes: error_logger.py, TextPreprocessing.py, TestTextPreprocessing.py, EvaluateModel.py, EvaluateClassifier.py and EnsembledModel.py.
- Trained XGBoost models on 2000 samples to compare the performance between the model trained on raw text embeddings and preprocessed text embeddings. Trained the XGBoost model 'xgb_grid_model' and the LightGBM 'lgb_clf_new' on the DAIG-V2 dataset for the Classifier.
- Got the 10k unseen text embeddings for model evaluation and evaluated all the models.

Ritika Kumar:

- Trained XGBoost model on a merged dataset of 504,731 data points with GridSearchCV to obtain best parameters. Trained AdaBoost and LightGBM model on merged dataset and obtained comparison between the model trained on raw text embeddings and preprocessed text embeddings.
- Code for module: TextEmbeddings.py, TestTextEmbeddings.py, Classifier.py, run_test.py, whowrotethis_app.py, UserApp.py
- Generated test embeddings for verification: embeddings_gpt_2.npy, embeddings_bert.npy, embeddings_instructor_xl.npy, embeddings_longformer.npy

Ardavan Mehdizadeh:

- HPC cluster computing for calculations; NEU Discovery HPC, 4 Nodes in the d2r2 exclusive partition, x86_64 architecture with 512 CPUs; Linux 3.10.0 with 500 GB of RAM; Running: Instructor-Text Embedding and GPT-2 text embedding models; XGBoost Model. Trained SVM model (performed poor on AIs); Plotly for plotting and evaluation of models.

Resources:

1. Datasets: AI vs Human Text (Link); DAIG-V2 (Link); LLM:7 prompt training dataset (Link); LLM: Mistral-7B Instruct texts (Link).
2. Packages: Pandas; Numpy; Scikit-Learn; XGBoost; NLTK; Tensorflow 2.0; Transformers; InstructorEmbedding; Pickle; Sentence-transformers == 2.2.2; Torch; Matplotlib; streamlit.