DDA4210 Report

Liu, Juntao
Tan, Zhaohong
Yang, Jinglan
Zhao, Rui

May 11, 2024

Abstract

This report describes the development and evaluation of classifiers for detecting and classifying AI-generated and human-generated text. Using integrated methods and feature analysis, through extensive experiments, we identified key findings about classifier performance, feature importance, and model diversity. Overall, this research helps maintain academic integrity as well as the integrity of information on online platforms in the face of the proliferation of AI-generated content.

1 Introduction

Large Language Models (LLMs) is one of the core research areas in artificial intelligence. In recent years, with the rapid popularization of Large Language models such as ChatGPT and Wenxinyiyan, Large Language Model (LLM) has been widely used in work and daily life, bringing a lot of convenience to people's production and life. However, the wide application of large language models also brings risks and hazards of abuse. For example, some users use ChatGPT to complete academic reports, criminals use ChatGPT to create false news to spread rumors, etc. The spread of false information, copyright infringement, academic misconduct, cheating and phishing attacks have endangered normal human society and adversely affected social harmony. And, with the development of large-scale language models, AI-generated texts are more difficult to distinguish from those actually constructed by humans, and it has become difficult for humans to accurately determine whether a text is AI text or not. In this context, AIGT (Artificial Intelligence-generated text) detection is an effective solution. To this end, this project aims to find a way to correctly classify human text and AI text in English context.

2 Literature review

At present, there are several research focuses on detecting AI-generated texts and the mainstream text classifiers are mainly implemented based on the following principles. "There are existing detection methods, such as feature-based classifiers, methods that differentiate human and AI text using statistical measures, along with methods that use fine-tuned language models to classify text as human or AI.", mentioned by Bhattacharjee and Liu (2023). And there are several classes of AI text detectors such as watermarking-based, neural network-based, zero-shot, and retrieval-based detectors.

To ensure text authenticity in various applications, AI text classifier tools have become a reliance on distinguishing between human-written and AI-generated content. According to M. Elkhatat, Elsaid, Almeer (2023), "For instance, OpenAI, which developed ChatGPT, introduced an AI text classifier that assists users in determining whether an essay was authored by a human or generated by AI. This classifier categorizes documents into five levels based on the likelihood of being AI-generated:very unlikely, unlikely, unclear, possibly, and likely AI-generated."“Other AI text classifier tools include Writer.com's AI content detector, which offers a limited application programming interface API-based solution for detecting AI-generated content and emphasizes its suitability for content marketing. Copyleaks, an AI content detection solution, claims a 99% accuracy rate and provides integration with many Learning Management Systems (LMS) and APIs. GPTZero, developed by Edward Tian, is an Open AI classifier tool targeting educational institutions to combat AI plagiarism by detecting AI-generated text in student assignments. Lastly, CrossPlag's AI content detector employs machine learning algorithms and natural language processing techniques to precisely predict a text's origin, drawing on patterns and characteristics identified from an extensive human and AI-generated content dataset."

The existing detectors have uneven performance in the face of texts of different languages, contexts, themes and lengths. Moreover, research by Sadasivan et al. (2023) supports that spoofing attacks on various AI text detectors by using recursive interpretation may cause the classifier to mistakenly detect human text as AI text, causing adverse effects.[3] In short, with the advent of newer, larger, and more powerful LLMS, the challenges of AI text detection will continue to intensify.

3 Methodology

3.1 Data preprocessing and feature engineering

Our data comes from a file called `processed_data.csv`, which contains 29,133 English texts with an average word count of about 200 words. Of these, 17,508 texts were generated by students (humans) and 11,625 by AI, and both were labeled with 0 and 1, respectively.

3.1.1 Introduction of attention mechanism

Why do we need an attention mechanism?

Attention mechanisms can be viewed as a kind of bionic design, a machine that mimics the attentional behavior of a human reading, listening and speaking. Because when the human brain performs a reading task, it is not a strictly decoding process, but something close to pattern recognition. The meaning of the same word changes depending on the topic of the conversation, so the brain automatically ignores low probability, low value information. The attention mechanism is implemented by adjusting the direction of attention and weighting the model according to the specific task objectives. In other words, in the hidden layer of the neural network, the weight of the attention mechanism is increased so that content that does not fit the attention model is weakened or forgotten.

Specific principles implementation:

Take the self-attention mechanism as an example. Suppose that we have a sentence in which each word is represented by x_i, the new vector a_i is obtained by Embedding on this paragraph, where W represents the parameter matrix of the Embedding.

$a_i = \mathrm{W}x_i$

a_i will serve as input data for the attention mechanism, and each $a_i$will be multiplied by three matrices, namely q,k, and v, which will be shared throughout the process.

$q_i = W_q a_i$, $k_i = W_k a_i$, $v_i = W_v a_i$

In general, the meaning of q(Query)is used to match other words, more accurately, it is used to calculate the correlation or relationship between the current word or word and other words or words. The meaning of k(Key)is used to match q, and can also be understood as a word or key information of a word. After the dot product of q and k, you get scaled dot-product attention, where d represents the matrix dimensions of q and k:

$e_{1,i} = q_1 k_i / \sqrt{d}$

Then, it is to perform softmax operation on it to obtain $\alpha_{1,i}$:

$\alpha_{1,i} = \exp(e_{1,i}) / \sum_{j=1}^{n} \exp(e_{1,j})$

The meaning of v mainly represents the important information representation of the current word or word, and can also be understood as the important features of the word. In the v operation, the $\alpha_{1,i}$, and v_i obtained after the q and k operations are multiplied respectively to obtain the attention b1 that can consider the entire sequence:

$b_1 = \sum_{i=1}^{n} \alpha_{1,i} v_i$

The multi-head attention mechanism used in this project is a variant developed on the basis of the self-attention mechanism to enhance the expressiveness and generalization ability of the model.

The multi-head attention mechanism is that after a_i is multiplied by a q,k,v, it

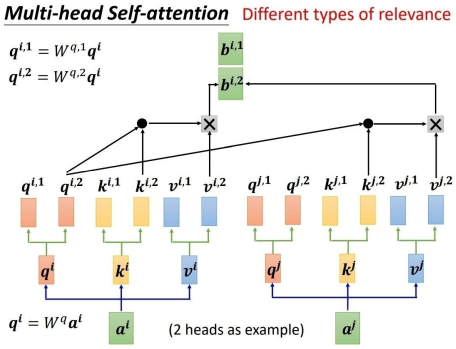

Figure 1: Multi-head attention mechanism

will assign multiple q,k,v. It achieves a richer representation by using multiple independent attention heads, calculating the attention weights separately, and concatenating or weighted summing their results.

3.1.2 Code implementation

The code to preprocessing the text contains three functions: `data_input`, `embedding`, and `multihead_attention`.

`data_input function`:

The main purpose of this function is to prepare data for the training of the model. Specifically, this function performs the following steps: Load the data from a CSV file named `processed_data.csv`, which is located in the same directory as the script. Shuffle the order of the data and reset the index. Convert the first n lines in the CSV file to integers and use those integers to create a PyTorch tensor containing the sample label. Build a vocabulary with the number of occurrences of each word in the text data set. Create a word-to-index mapping, where each word corresponds to an integer index, and map the most common words to the smallest index value. Converts the text to an indexed sequence and populates the text to the same length for easy model training.

`embedding function`:

This function is used to embed words into dense vectors. Specifically, this function performs the following steps: Gets the size of the vocabulary and specifies the dimensions of the embedding, mapping words as dense vectors via nn.Embedding. Use torch.mean to average pool the embedded input to get the representation vector of the sentence.

`multihead_attention function`:

This function is used to perform multi-head self-attention calculations. Specifically, this function performs the following steps: Examine the dimensions of the input tensor and take the last dimension of the input tensor as the input dimension. Instantiate an `nn.MultiheadAttention` module. The output tensor

of attention is obtained by using `nn.MultiheadAttention` for self-attention calculation of input. The main function of this code is to prepare and process text data, and it performs word embedding and self-attention calculations. These features are very important steps when doing natural language processing (NLP) tasks.

3.2 Model selection

We chose Logistic Regression, KNN, Random Forest as basic models of the classifier. A stacking method with KNN and Random Forest as the base learner and Logistic Regression as the meta-learner is applied to integrate the effects of the learner in pursuit of better and more robust prediction results.

3.2.1 Brief overview of Logistic Regression, KNN, Random Forest, and CNN

Logistic Regression:

Type: Supervised learning algorithm used for binary classification tasks. Functionality: Estimates the probability that an instance belongs to a particular class (e.g., AI-generated vs. human-generated) based on input features.

Key Features: Utilizes the logistic function (sigmoid) to transform the output into a probability score. Learns weights for input features to make predictions.

Advantages: Simple and interpretable. Efficient for large datasets. Limitations: Limited to linear decision boundaries. Assumes features are independent.

K-Nearest Neighbors (KNN):

Type: Instance-based, non-parametric learning algorithm. Functionality: Classifies data points based on the majority class among their k-nearest neighbors in feature space.

Key Features: No explicit training phase; all training data points are stored. Classification decision based on proximity in feature space. Advantages: Simple and intuitive concept. Can adapt well to complex decision boundaries.

Limitations: Computationally expensive during inference, especially with large datasets. Sensitive to the choice of k.

Random Forest:

Type: Ensemble learning method based on decision trees. Functionality: Constructs multiple decision trees during training and outputs the class that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees.

Key Features: Each tree is built on a random subset of features and training data. Combines predictions from multiple trees to reduce overfitting and improve robustness.

Advantages: Robust against overfitting. Provides feature importance ranking.

Effective for high-dimensional data. Limitations: Less interpretable compared to single decision trees. Requires careful tuning of hyperparameters.

Convolutional Neural Network (CNN):

Type: Deep learning architecture designed for processing structured grid-like data, such as images and sequences.

Functionality: Employs multiple layers of convolutions and pooling operations to automatically learn hierarchical representations of data. Key Features: Convolutional layers extract spatial patterns from input data. Pooling layers down sample feature maps to reduce dimensionality. Typically followed by fully connected layers for classification.

Advantages: State-of-the-art performance for image and sequential data analysis. Learns hierarchical features automatically.

Limitations: Requires large amounts of training data and computational resources. Prone to overfitting without proper regularization techniques.

3.3 Code implementation

Three functions `Logistic_Regression`, `KNN` and `randomforest` are defined in the code, which are used to realize the training and prediction functions of logistic regression, KNN and random forest classifier respectively. There is also a function called `stacking`, which implements a stacking ensemble learning method. And they return the accuracy of the training set and the test set, as well as the prediction of the training set and the test set.

In building the CNN, the `Sequential` model in the `Keras` library is first used to initialize the model, and the different hierarchies of the model are defined. Next, the model is compiled using the `compile` method, specifying the loss function, optimizer, and evaluation criteria. The CNN model is trained using fit method, which specifies the proportion of training data, training labels, training rounds and evaluation sets in the training set, and adds `TensorBoard` callbacks to monitor the training process. The `evaluate` method is used to test and evaluate the model, and the `predict` method is used to obtain the classification result of the test data. Finally, the prediction accuracy of the model is calculated and the result is output.

4 Experiment Design

We apply a cross-validation loop to evaluate the performance of different classifiers. `fold` is a K-fold cross-validation iterator, and the split method returns the index of the training and test sets for each partition. `train_x` and `test_x` contain the sample features corresponding to the corresponding index, and `train_y` and `test_y` contain the sample labels corresponding to the corresponding index. The classification function is called to train and predict the random forest classifier on the divided training set and test set, and the accuracy and prediction results are obtained. Use the `confusion_matrix` function to calculate the confusion

matrix and calculate the accuracy, recall, and F1 values for the training set. Repeat the above steps for the test set. Next, the accuracy, recall, and F1 values for the average training and test sets are printed, and the results are saved to the corresponding list. To achieve visualization, we use the `matplotlib` library to plot the accuracy, recall and F1 value curves of the training and test sets at different depths and save the images.

5 Results

5.1 Presentation of classification results

The classification results and performance of different models are shown below, including precision, recall rate and f1 score under different sample sizes. By comparison, we can see that random forest and the stacking classifier have good performance. As the sample size increases to 10000, the accuracy rate can be stable at about 0.75, which is higher than the baseline model accuracy rate of 0.68.

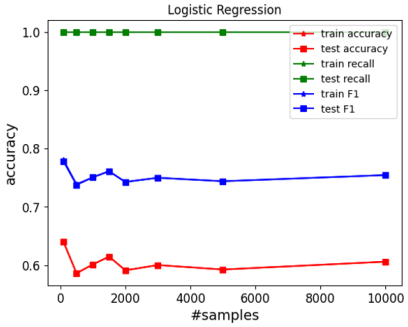

Figure 2: Logistic regression

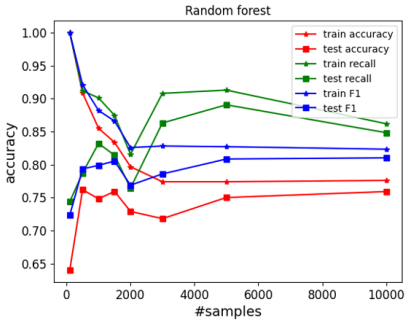

Figure 3: Random forest

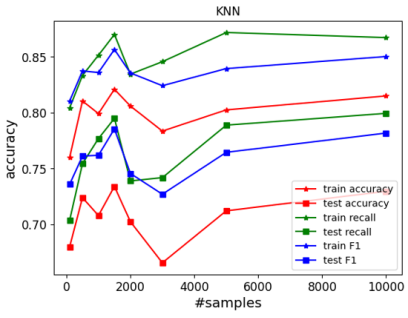

Figure 4: KNN

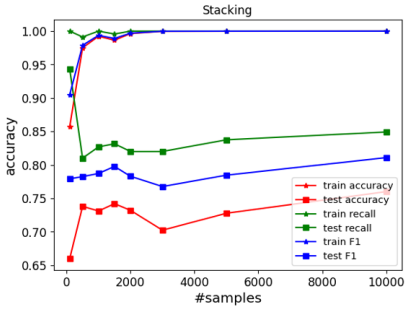

Figure 5: Stacking

5.2 Interpretation of results

Random forest is known for its robustness against over fitting, this robustness could lead to better generalization performance on unseen data compared to other models. Random forest provides a feature importance ranking, which can help identify the most informative features for classification. Benefiting from the diversity of the underlying models, for example, KNN may capture local patterns while random forests can handle nonlinear relationships, the stacking classifier gains better overall performance by combining these different predictions.

5.3 Limitations

The size of the data set is small. Over fitting can occur when more complex models are applied. CNN's attention and explainability are weak. Our model is only applicable in the English context, and robustness issues may arise when facing other languages such as Chinese.

6 Conclusion

6.1 Summary of Key Findings

In this project, we developed and trained text classifiers to classify AI-generated text. Experiments and evaluations reveal the differences between different machine learning algorithms and techniques in handling this challenging task, showing that the integrated approach and meta-integration are effective in capturing the complexity of AI-generated text. The significance of this project: A high-precision classifier for detecting AI-generated text can effectively detect academic plagiarism by students using AI to write papers, while enhancing content moderation strategies on digital platforms. Our findings help address the challenges posed by the proliferation of AI-generated content, reducing the spread of harmful or deceptive content.

6.2 Suggestions for future work

Future research could explore more advanced machine learning techniques to further improve classification accuracy and robustness, providing additional capabilities for capturing subtle patterns in AI-generated text. The evaluation of the classifier is extended to different text sources and domains, and its generalization ability and robustness in different contexts are deeply understood. Evaluating the performance of real data sets from various sources can verify the effectiveness of the classifier in real-world scenarios.

References

[1] Bhattacharjee, A. & Liu, H (2023). Fighting Fire with Fire: Can ChatGPT Detect AI-generated Text? SIGKDD Explorations, 25(2), 14-21.

[2] Elkhatat A. M., Elsaid K. & Almeer S. (2023). Evaluating the efficacy of AI content detection tools in differentiating between human and AI-generated text. International Journal for Educational Integrity, 19(1), 1-16.

[3] Sadasivan, V. S., Kumar, A., Balasubramanian, S., Wang, W., & Feizi, S. (2024, February 19). Can ai-generated text be reliably detected?. arXiv.org.