Transactions on Computer Science and Intelligent Systems Research
ISSN: 2960-1800, eISSN: 2960-2238 | Volume 5
AIDML 2024

Exploiting Machine Learning Model Ensemble for AI-Generated Texts Detection

Chucheng Zhou

School of Computing and Data Science, Xiamen University Malaysia, Sepang, Selangor, 43900, Malaysia

SWE2209553@xmu.edu.my

Abstract. In recent years, the development of Large Language Models (LLMs) has reached a state of maturity, resulting in texts that are becoming increasingly indistinguishable from those written by humans. This progress has sparked a growing need for precise and efficient methods to detect Artificial Intelligence (AI)-generated texts, as the blend of human and machine authorship becomes more seamless. This study utilizes data from student-written papers and articles generated by various LLMs to develop a machine learning model capable of accurately distinguishing whether an article was written by a student or an LLM. Four different classification models (MultinomialNB, SGDClassifier, LGBMClassifier, and CatBoostClassifier) and their ensemble models with weighted combinations were chosen to detect AI-generated texts. To evaluate the effectiveness of these models, the Area Under Curve (AUC) score metric was employed. The results indicate that the CatBoostClassifier model performs the best in mitigating overfitting, while the ensemble model demonstrates the optimal predictive performance. This discovery holds significant importance for enhancing the accuracy of detecting AI-generated articles.

Keywords: Large language models; AI-generated texts; ensemble model.

1. Introduction

With the widespread adoption of Large Language Models (LLMs), many individuals are using them to replace or modify tasks traditionally performed by humans [1]. Educators, in particular, are concerned about their impact on students' skill development, although many remain optimistic that LLMs will eventually become useful tools to enhance students' writing abilities. However, there are inevitable challenges, such as the potential encouragement of plagiarism [2]. LLMs undergo training on extensive datasets encompassing both text and code, empowering them to produce text closely resembling human-generated content. For instance, students may use LLMs to produce content that is not their own, resulting in the omission of crucial learning points. Additionally, in the academic realm, individuals might exploit large language models to generate deceptive and inaccurately misleading articles, especially in situations where there is insufficient reference material. This can mislead readers and propagate incorrect viewpoints.

Therefore, detecting Artificial Intelligence (AI)-generated text can assist teachers in distinguishing between student and plagiarized content [3]. It can also aid paper reviewers in discerning AI-generated content from manually written content to prevent deception. In recent years, scholars have extensively explored the detection of AI-generated text, yielding rich research outcomes. However, there is a lack of comprehensive or in-depth comparative analysis among different approaches.

Hence, this paper takes a series of articles written by students and those generated by various LLMs as examples. This work employs four models and their ensemble models to detect whether the text is AI-generated. By utilizing the Area Under Curve (AUC) metric to evaluate the models, the aim is to select the best predictive model. This work intends to provide a rational reference for educational institutions and the academic industry in determining whether text is generated by AI.

2. Method

2.1. Multinomial Naive Bayes (NB)

The MultinomialNB algorithm is primarily suitable for computing probabilities of discrete features and is commonly applied in document classification [4]. It can determine the probability that a document falls into specific categories, assigning the category with the highest probability to the document. In the experiment aimed at discerning whether a text falls into the AI or student category, an evaluation of the magnitudes of $P(AI|Text)$ and $P(Student|Text)$ is necessary. Given that the text essentially consists of individual keyword features, it becomes imperative to compute $P(AI|word1, word2, word3, \dots)$ and $P(Student|word1, word2, word3, \dots)$. In instances involving multiple conditions for one category, the Naive Bayes formula is employed:

$$P(C|W) = \frac{P(W|C)P(C)}{P(W)} \tag{1}$$

Here, W represents the features of a given document, which are the different words extracted from the article. C is the category of the document. As there is a necessity to extract keywords from the document, the formula can be interpreted as follows:

$$P(C|F1, F2, \dots) = \frac{P(F1,F2,\dots|C)P(C)}{P(F1,F2,\dots)} \tag{2}$$

$$P(F1|C) = \frac{N_i}{N} \tag{3}$$

Here, $P(C)$ is the probability of each document category. $P(F1, F2, \dots|C)$ is the probability of features given a category. The features in this context are the words predicted to appear in the document. $F1$ represents a specific word predicted to appear in the document, N_i is the number of times $F1$ appears in all documents of category C, and N is the total number of occurrences of all words in documents belonging to category C. Since the denominator is the same, $P(F1, F2, \dots|C)$ and $P(C)$ are the only values which need to compare.

2.2. Stochastic Gradient Descent (SGD) Classifier

The SGDClassifier utilizes stochastic gradient descent to minimize the selected loss function [5]. During the training process, it calculates gradients and updates model parameters, gradually adjusting the model to minimize the loss function. This process iterates continuously until reaching a predetermined number of training rounds or other stopping conditions. In this experiment, modified_huber is used as the loss function for this model, which is defined as follows:

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(\hat{y} - y)^2, & for\ |\hat{y} - y| \le \delta \\ \delta\left(|\hat{y} - y| - \frac{1}{2}\delta\right), & otherwise \end{cases} \tag{4}$$

Here, y represents the actual label, $\hat{y}$ is the model's predicted value, and δ is a user-defined threshold. The design of this loss function incorporates squared loss when the prediction is close to the actual label and linear loss when the prediction deviates significantly. This loss function performs relatively well when handling outliers. As it employs a modified version of the Huber loss function, it handles absolute differences between predicted and true values in a relatively mild manner, limiting the impact of outliers on the model.

2.3. Light Gradient Boosting Machine (LightGBM) Classifier

LightGBM is a machine learning model based on Gradient Boosting Decision Trees (GBDT) [6], designed to enhance training efficiency and model performance [7]. It employs the gradient boosting

algorithm, iteratively training multiple decision trees to construct a powerful predictive model. Technical innovations include the use of Gradient-based One-Side Sampling (GOSS) to significantly reduce the estimation scale of small gradient data instances and the adoption of Exclusive Feature Bundling (EFB) to effectively reduce the number of features [7]. LightGBM places a strong emphasis on performance, rendering it especially well-suited for handling extensive datasets and feature spaces with high dimensions. It utilizes histogram-based algorithms and parallel computation to improve training speed and memory utilization efficiency. In this experiment, cross_entropy is used, suitable for binary classification problems, as the loss function for the LightGBM classifier. By setting the loss function to cross-entropy, the model aims to minimize cross-entropy loss during the training process, thereby improving fitting to the training data.

2.4. CatBoost Classifier

Similar to LightGBM, CatBoost is an enhanced implementation within the GBDT algorithm framework [8]. CatBoost, a GBDT framework built on oblivious trees, boasts a streamlined set of parameters, robust support for categorical variables, and impressive accuracy. Its primary emphasis lies in efficiently managing categorical features. Furthermore, CatBoost tackles challenges like Gradient Bias and Prediction Shift within the GBDT framework, diminishing the risk of overfitting and enhancing the algorithm's precision and generalization capabilities.

In contrast to LightGBM, CatBoost introduces a novel approach for automatically converting categorical features into numerical ones. Initially, it conducts a statistical analysis on categorical features, determining the frequency of each category, and subsequently introduces hyperparameters to generate new numerical features. CatBoost also harnesses the amalgamation of categorical features, leveraging the interconnections between them to significantly enhance the feature space. It employs a ranking boosting technique to counteract noise in the training set, averting bias in gradient estimation and ultimately addressing the predicament of prediction shift.

3. Experiments and Results

3.1. Experimental Settings

A classification model will be constructed based on the DAIGT V2 Train Dataset to ascertain whether the text is generated by AI [9].

Initially, redundant data will be filtered out through similarity-based criteria. Subsequently, the BPETrainer will be employed to pre-train a tokenizer on the test set text. Both the training and test sets will be tokenized using the acquired tokenizer, and statistical features will be derived from the consistent vocabulary. Following tokenization, Term Frequency-Inverse Document Frequency (TFIDF) values of N-grams (3,5) in the text will serve as feature vectors [10]. These features will be individually input into classifiers, including MultinomialNB, SGDClassifier, LGBMClassifier, CatBoostClassifier, and an ensemble model for training purposes. Detection will be executed on the test set, and, ultimately, AUC evaluation metric scores will be obtained.

3.1.1. Dataset

The experimental dataset includes not only 1375 samples of persuasive essays written by students and 3 samples of persuasive essays generated by LLMs provided by the Kaggle website but also external data from the DAIGT V2 Train Dataset, as shown in Figure 1. The persuade_corpus consists of 25,996 publicly available samples of persuasive essays written by students, while the remaining samples, totaling 19,509, are articles generated by various LLMs. The detailed information about the dataset is shown in Table 1.

Table 1. Information about the dataset.

Sample Source	Sample quantities
persuade_corpus	25996
chat_gpt_moth	2421
llama2_chat	2421
mistral7binstruct_v2	2421
mistral7binstruct_v1	2421
original_moth	2421
train_essays	1378
llama_70b_v1	1172
falcon_180b_v1	1055
darragh_claude_v7	1000
darragh_claude_v6	1000
radek_500	500
NousResearch/Llama-2-7b-chat-hf	400
mistralai/Mistral-7B-Instruct-v0.1	400
cohere-command	350
palm-text-bison1	349
radekgpt4	200

3.1.2. Evaluation Metrics

Regarding evaluation metrics, this paper employs the Area Under the ROC Curve (AUC) score to evaluate the performance of binary classification models. As depicted in the figure below, the AUC score varies from 0 to 1. A perfect model achieves an AUC score of 1, indicating impeccable discrimination ability to distinguish all positives and negatives. An AUC value of 0.5 signifies that the model lacks discriminatory ability, akin to random guessing. Generally, the closer the AUC value is to 1, the better the model's performance, while a lower AUC value implies poorer performance.

The ROC curve serves as a graphical tool for assessing the performance of binary classification models. It illustrates the True Positive Rate (TPR) on the vertical axis and the False Positive Rate (FPR) on the horizontal axis, showcasing the model's performance at different classification thresholds. The formulas for TPR and FPR calculation are as follows:

$$TPR = \frac{TP}{TP+FN} \tag{5}$$

$$FPR = \frac{FP}{FP+TN} \tag{6}$$

, where TP denotes True Positives, indicating the count of samples accurately identified as positive. FN refers to False Negatives, representing the instances where positive samples were inaccurately classified as negative. FP stands for False Positives, signifying the occurrences where negative samples were incorrectly classified as positive. Lastly, TN represents True Negatives, denoting the number of samples correctly identified as negative.

3.2. Data Preprocessing

Upon importing the dataset, the first step involves excluding rows with a similarity exceeding 0.92, followed by reindexing the dataset. Subsequently, text processing is set to be case-sensitive, and the vocabulary size is defined as 30,522, indicating the use of a model with a vocabulary comprising 30,522 distinct tokens.

In natural language processing, a Tokenizer is used to split natural language text into different lexical units (tokens). This process typically involves segmenting words, punctuation, numbers, etc., and removing some noise or redundant information. The result of Tokenization is a sequence containing

several lexical units, which can be used to construct a vector representation of the text. For example, the text "Hello, World!" after tokenization might be split into "Hello##,##World##!" and then encoded as a vector [21,30,123,5].

Compared to traditional methods that tokenize based on characters, BPE (Byte Pair Encoding) tokenizer's core idea is to iteratively merge the most frequent byte pairs (or character pairs) to reduce the total number of bytes (or characters) in the entire dataset. This approach is particularly useful for handling Out-Of-Vocabulary (OOV) issues, improving the model's generalization ability to rare or new words, and reducing the risk of semantic gaps.

In this experiment, this paper used the BPETrainer to train the BPE tokenizer. The training process is as follows:

Step 1. Initialize the vocabulary: BPETrainer first breaks down all words in the dataset into base characters (such as letters or Chinese characters) plus a special end symbol. It then counts and initializes a base vocabulary, including all unique characters and their frequencies.

Step 2. Iterative merging: BPETrainer iteratively performs the following steps:

• Based on the current vocabulary, it counts the occurrences of all adjacent character pairs (bigrams).

• Identifies the character pair with the highest frequency, merges this pair into a new symbol (i.e., the combination of these two characters), and adds it to the vocabulary.

• Updates the words in the dataset, replacing the selected character pair with the corresponding new symbol.

Step 3. Repeat iteration: Repeat the iterative process of Step 2 until reaching the preset vocabulary size or merging times.

Step 4. Generate the model: After completing all iterations, BPETrainer generates the BPE tokenizer based on the final vocabulary.

After training the tokenizer, the text in the dataset can be transformed into feature vectors consisting of numerical identifiers.

3.3. Feature Engineering

Subsequently, two methods will be employed to extract valuable features from the feature vectors.

Initially, the N-Gram method will be utilized to generate a list of grams. This algorithm, rooted in statistical language models, operates by executing a sliding window operation on the text content in byte-sized chunks of size N. This process forms a sequence of byte fragments, each of length N, termed a gram. The frequencies of all grams are tallied, and after filtering based on a predetermined threshold, a pivotal gram vocabulary is established, serving as the vector feature space for the text. Each gram in the list becomes a feature vector dimension. In this experiment, the minimum word frequency is set to 2, necessitating each gram to appear at least twice in its respective text. Subsequently, the N-Gram method is employed to extract grams from the test set text within an N range of 3-5, thereby forming a comprehensive gram vocabulary.

Following this, the TF-IDF values for each gram in the gram vocabulary are computed using the TF-IDF method. (For grams absent in a text, the TF-IDF value is 0; for those present, it is the TF-IDF value calculated for that specific gram in that text.) Subsequently, each text in the input corpus is vectorized to ensure consistent dimensions across all texts. The TF-IDF method is elucidated below, wherein TF signifies Term Frequency, representing the number of times a word appears in a document. It is evident that words appearing frequently in an article hold significance. However, commonly occurring words in text statistics, such as 'we' and 'are', may interfere with statistical analysis. To address this, methods like a stop-word corpus are employed to eliminate these insignificant words. Based on word frequency, an "importance" weight is assigned to each word in the test set. The most common words in the test set receive the smallest weight, less common words

receive smaller weights, and less frequent words receive larger weights. This weight is termed "Inverse Document Frequency" (IDF). Ultimately, the TF-IDF value for a word is determined by the multiplication of these two factors. The greater a word's significance in the article, the higher its corresponding TF-IDF value. In this experiment, the higher the TF-IDF value of a gram, the more crucial it is for that specific text. Following the extraction of features through the aforementioned methods, the desired feature vectors are acquired. Subsequently, the preprocessed training set texts are trained using MultinomialNB, SGDClassifier, LGBMClassifier, CatBoostClassifier, and ensemble models, and the experimental results are tested on the test set texts.

3.4. Experimental Details

The implementation of both SGDClassifier and MultinomialNB models was carried out using the Python library scikit-learn, while the LGBMClassifier was implemented from the lightgbm library, and the CatBoostClassifier from the catboost library. As the test data lacked text categories, the original training data is partitioned into a 6:4 ratio, creating a training set and a test set. For SGDClassifier, a maximum of 8000 iterations was specified, and the algorithm halts when the change in the loss function is less than 0.0001. Regarding MultinomialNB, the Laplace smoothing parameter alpha was set to 0.02. In the case of LGBMClassifier, the number of iterations was set to 2500, with colsample_bytree at 0.78, colsample_bynode at 0.8, min_data_in_leaf at 115, max_depth at 23, and max_bin at 898. A carefully tuned learning rate of approximately 0.0058, L1 regularization weight around 4.56, and L2 regularization weight around 2.97 were utilized. For CatBoostClassifier, the number of iterations was set to 2500, subsample to 0.4, learning rate to approximately 0.0056, and L2 regularization weight to about 6.66. Constant labels were permitted to handle situations where all labels in the training set share the same value. Lastly, an ensemble of the four classifiers was performed, with weights assigned as follows: CatBoostClassifier weight set to 0.4, SGDClassifier weight set to 0.3, MultinomialNB weight set to 0.2, and LGBMClassifier weight set to 0.1.

3.5. Results

Table 2 contains the performance results on the training set, while Table 3 presents the results on the test set. It is noteworthy that CatBoostClassifier and the ensemble model exhibit excellent performance. The ensemble model achieved a remarkable AUC score of 0.969 on the training set and maintained a 0.900 AUC score on the test set, demonstrating superior performance without overfitting.

Various classification models were also compared. Evidently, SGDClassifier and MultinomialNB demonstrated higher scores on the training set, with scores of 0.941 and 0.926, respectively, whereas CatBoostClassifier exhibited a lower score of 0.867. However, on the test set, the CatBoostClassifier model outperformed both the SGDClassifier and MultinomialNB models. This underscores the efficacy of the CatBoostClassifier model in mitigating overfitting. In the case of LGBMClassifier, initial challenges with overfitting resulted in unsatisfactory outcomes. To tackle this issue, adjustments were made to the learning rate and the number of iterations, leading to improved performance.

Table 2. Results on the training set.

Model	AUC
MultinomialNB	0.926
SGDClassifier	0.941
LGBMClassifier	0.890
CatBoostClassifier	0.867
Ensemble Model	0.969

Table 3. Results on the testing set.

Model	AUC
MultinomialNB	0.820
SGDClassifier	0.824
LGBMClassifier	0.851
CatBoostClassifier	0.876
Ensemble Model	0.900

From the experiments, this paper can draw the following two conclusions:

Firstly, Weighted ensemble of MultinomialNB, SGDClassifier, LGBMClassifier, and CatBoostClassifier models performs better than individual models. This ensemble model can be utilized to predict whether an article is generated by AI. The improved performance of the ensemble model is attributed to the diverse learning algorithms and strategies employed by each model, leveraging their individual strengths and weaknesses. In this experiment, MultinomialNB is suitable for text classification, SGDClassifier uses stochastic gradient descent for training, LGBMClassifier is a gradient boosting tree model, and CatBoostClassifier is an ensemble learning model based on gradient boosting. This combination allows for a more comprehensive consideration of the diversity in the data.

Secondly, CatBoostClassifier performs better in mitigating overfitting compared to other classifiers. In scenarios with large and diverse datasets, it is advisable to appropriately increase the weight of CatBoostClassifier in the ensemble model. The superior performance of CatBoostClassifier can be attributed to its use of adaptive learning rates, automatically adjusting the learning rate based on the performance of each tree. This feature helps the model adapt better to the data, reducing the risk of overfitting. Additionally, CatBoost is specifically designed to handle categorical features without the need for one-hot encoding or other complex preprocessing. This prevents the introduction of excessive noise when dealing with categorical data, aiding better generalization to unseen data. Finally, CatBoost itself is an ensemble learning algorithm, combining predictions from multiple tree models. This ensemble characteristic helps reduce the risk of overfitting in individual models and enhances overall generalization performance.

4. Conclusion

This paper concentrates on the pivotal task of discerning whether articles are authored by AI, a matter of significant practical importance within the education industry. Experiments were conducted using the DAIGT V2 Train Dataset paper dataset, involving vocabulary training with a BPE tokenizer and the extraction of TFIDF values for text N-grams (3,5) as features in the data preprocessing stage. Multiple models, including MultinomialNB, SGDClassifier, LGBMClassifier, and CatBoostClassifier, were explored, all yielding promising results. A noteworthy finding is that the weighted ensemble of MultinomialNB, SGDClassifier, LGBMClassifier, and CatBoostClassifier surpasses individual models, indicating substantial reference value in detecting AI-generated text. Additionally, CatBoostClassifier exhibits superior performance in mitigating overfitting compared to other classifiers, suggesting that in scenarios involving large and diverse datasets, particularly when predicting varied data types, assigning higher weight to the CatBoostClassifier model in the ensemble enhances prediction outcomes.

Looking ahead, there are several directions for further research and improvement. First, introducing more advanced feature engineering techniques and pre-classifying text could potentially enhance prediction accuracy. Second, exploring ensemble methods to leverage the strengths of various models may lead to more robust predictions. Finally, in future studies, researchers could delve into predicting the proportion of content that is manually written versus AI-generated, further enhancing the rigor of predictions.

In summary, the precise identification of AI-generated text holds paramount importance in maintaining fairness and rigor within the academic industry. This project presents a solution for forthcoming academic research dedicated to the detection of AI-generated text. Continued exploration and refinement are anticipated to address the increasing demand for text detection in educational institutions and the academic sector.

References

[1] Chang, Yupeng, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen et al. A survey on evaluation of large language models. ACM Transactions on Intelligent Systems and Technology, 2023: 1-43.

[2] Sadasivan, Vinu Sankar, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. Can AI-generated text be reliably detected?. ArXiv Preprint, 2023: 2303.11156.

[3] Shah, Aditya, Prateek Ranka, Urmi Dedhia, Shruti Prasad, Siddhi Muni, and Kiran Bhowmick. Detecting and Unmasking AI-Generated Texts through Explainable Artificial Intelligence using Stylistic Features. International Journal of Advanced Computer Science and Applications, 2023, 14(10): 1043-1053.

[4] Kibriya, Ashraf M., Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. Multinomial naive bayes for text categorization revisited. In AI 2004: Advances in Artificial Intelligence: 17th Australian Joint Conference on Artificial Intelligence, 2005, 17: 488-499.

[5] Kabir, Fasihul, Sabbir Siddique, Mohammed Rokibul Alam Kotwal, and Mohammad Nurul Huda. Bangla text document categorization using stochastic gradient descent (sgd) classifier. In 2015 international conference on cognitive computing and information processing, 2015: 1-4.

[6] Si, Si, Huan Zhang, S. Sathiya Keerthi, Dhruv Mahajan, Inderjit S. Dhillon, and Cho-Jui Hsieh. Gradient boosted decision trees for high dimensional sparse output. In International conference on machine learning, 2017: 3182-3190.

[7] Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 2017, 30: 1-9.

[8] Hancock, John T., and Taghi M. Khoshgoftaar. CatBoost for big data: an interdisciplinary review. Journal of big data, 2020, 7(1): 94.

[9] DAIGT V2 Training Dataset. URL: https://www.kaggle.com/datasets/thedrcat/daigt-v2-train-dataset. Last Accessed 2024/03/13.

[10] Christian, Hans, Mikhael Pramodana Agus, and Derwin Suhartono. Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF). ComTech: Computer, Mathematics and Engineering Applications, 2016, 7(4): 285-294.