# Detect AI-Generated Text: Identify which essay was written by a large language model

**Kristóf Benedek [1], Marcell Emmer[1], Izabell Anna Járó[2] and Lin Ruihan[3]**

[1] *Budapest University of Technology and Economics, TTK, Research Physicist MSc, 2nd year*
[2] *Budapest University of Technology and Economics, TTK, Medical Physics MSc, 2nd year*
[3] *Budapest University of Technology and Economics, KJK, Logistics Engineering MSc, 2nd year*

February 7, 2024

**In recent years, large language models (LLMs) have become increasingly sophisticated, capable of generating text that is difficult to distinguish from human-written text. In this project, we hope to foster open research and transparency on AI detection techniques applicable in the real world.**

## Important remark

Please find all our related code in this GitHub repository.

## 1 Introduction

Society is divided on whether artificial intelligence (AI) will ultimately be integrated into our daily lives to make it easier and benefit humanity or whether it will further increase the inequalities between the different social strata. When it comes to the large language models (LLM), a common concern besides the possibility that they can potentially replace humans, is related to their impact on students' skill development, and at the academic forefront, their potential to enable plagiarism.

LLMs are trained on a massive dataset of text and code, which means that they can generate text that is very similar to human-written text. For example, students could use LLMs to generate essays that are not their own, missing crucial learning keystones. Our work will try to help identify telltale LLM artifacts, investigate the current stage of LLM text detection, and further expand it. By using texts of moderate length on a variety of subjects and multiple, in some cases unknown, generative models, we aim to replicate typical detection scenarios and incentivize learning features that generalize across models. We will implement different types of categorizers, existing ones, and original ones, compare their performances and identify their potential weaknesses.

In our work, after some data analysis, we have made a pipeline for text augmentation with `Mistral-7B-Instruct-v0.1`. Then, the augmented dataset was used as a basis for training and testing our models. The work here has split into different approaches. One approach implemented some basic models and created an ensemble model out of them, the other tested different well-known, cheap text classifiers and implemented some with the help of shallow neural networks (SNN), deep neural networks such as convolutional (CNN), long short term memory (LSTM), bidirectional recurrent (BRNN), gated recurrent unit (GRU) and recurrent convolutional (RCNN). These were tested against each other with different train-test sets and they were taught with different feature engineering, tokenization/word vectorization methods if you wish, such as word embedding, TF-IDF, and count vectors. These methods were usually implemented on different levels, such as n-gram, word, or character level.

## 2 Large Language Models

Large language models (LLMs) are artificial intelligence algorithms that apply neural network techniques with lots of parameters to process and understand human languages or text using self-supervised learning techniques. Tasks like text generation, machine translation, summary writing, image generation from texts, machine coding, chat-bots, or Conversational AI are applications of the LLMs. Examples of such LLM models are Chat GPT by open AI, BERT (Bidirectional Encoder Representations from Transformers) by Google, etc.

LLMs are trained using deep learning techniques on massive amounts of text data and these models are capable of generating human-like text and performing various natural language processing tasks.

In contrast, the definition of a language model refers to the concept of assigning probabilities to sequences of words, based on the analysis of text corpora. A language model can be of varying complexity, from simple n-gram models to more sophisticated neural network models. However, the term "large language model" usually refers to models that use deep learning techniques and have a large number of parameters, which can range from millions to billions. These models can capture complex patterns in language and produce text that is often indistinguishable from that written by humans.

The architecture of an LLM primarily consists of multiple layers of neural networks, like recurrent layers, feedforward layers, embedding layers, and attention layers. These layers work together to process the input text and generate output predictions.

- The embedding layer converts each word in the input text into a high-dimensional vector representation. These embeddings capture semantic and syntactic information about the words and help the model to understand the context.
- The feedforward layers of Large Language Models have multiple fully connected layers that apply nonlinear transformations to the input embeddings. These layers help the model learn higher-level abstractions from the input text.
- The recurrent layers of LLMs are designed to interpret information from the input text in sequence. These layers maintain a hidden state that is updated at each time step, allowing the model to capture the dependencies between words in a sentence.
- The attention mechanism is another important part of LLMs, which allows the model to focus selectively on different parts of the input text. This mechanism helps the model attend to the input text's most relevant parts and generate more accurate predictions.

In our task, we had a much simpler job than in the case of these high-complexity, generative AIs, since we only had to classify the result of these models: we only had to decide whether the label of a given text is $0$ or $1$, i.e. corresponds to human-written text or AI-written one. We also didn't have to construct LLMs from scratch, not even for the text augmentation, since we could access pre-trained and freely available APIs through Kaggle and/or Hugging Face.

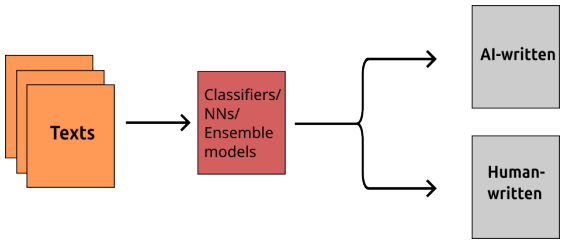


**Figure 1:** *Visualizing the workflow of the text classification.*

## 3 Data Augmentation

Data augmentation plays a critical role in enhancing the performance and reliability of models designed to detect AI-generated essays. In this section, we detail the process employed to generate synthetic essays using the Mistral-7B-Instruct model. This approach aids in creating a robust dataset, which is crucial for training and evaluating our detection algorithms.

### 3.1 Environment Setup

The model in focus, `Mistral-7B-Instruct-v0.1` from mistralai, a variant of the Generative Pre-trained Transformer optimized for instructional tasks, is initialized. This step involves loading both the tokenizer, which prepares input text for the model, and the causal language model responsible for text generation.

To reduce memory usage and speed up computation, the model uses `bfloat16` precision. The device allocation for the model's layers is set to automatic (`device_map='auto'`), enabling optimal use of available computational resources, i.e. the 2 T4 GPUs that Kaggle provides for free.

### 3.2 Data Augmentation Methodology

Key to the data generation process is the preparation of prompts that guide the AI in producing specific types of essays. These prompts are designed to ensure the generated data closely aligns with the characteristics of human-written academic essays. The competition dataset had $2$ example training prompts with instructions detailing the task. We wanted to create a dataset that is an extension of the popular DAIGT V2 Train Dataset which includes 5 other topics. The instructions for the extra topics were generated by ChatGPT-3.5.

Utilizing the prepared prompts, the Mistral-7B-Instruct model generates a diverse set of synthetic essays. This step involves finely tuning parameters such

as generation length and creativity controls to produce realistic and varied textual outputs. This includes randomly generating a grade level for the essay we want to generate, the probability of either passing the instructions to the model or not, and the probability of having typos in the generated essays up to the desired student grade level. The 400 essays generated this way were appended to the DAIGT V2 Train Dataset.

*The generation of synthetic data not only enriches the dataset but also provides a means to rigorously test and refine the detection algorithms, ensuring higher accuracy and reliability in real-world applications.*

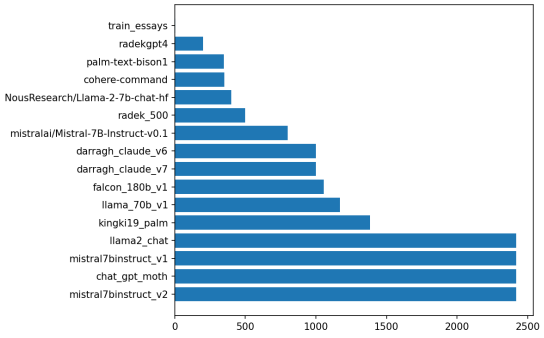


**Figure 3:** *The essays' number written by different types of AI models*

# 4 Text classification with different classifiers and neural networks

## 4.1 Exploratory Data Analysis

We used as dataset the `DAIGT_V2 dataset`[1] and augmented it with our pipeline, by adding 400 more essays to the existing over 44 800, resulting in a total 45 268 texts. Checking the class distribution, we found that even in the augmented data we have slightly more human-written essays:

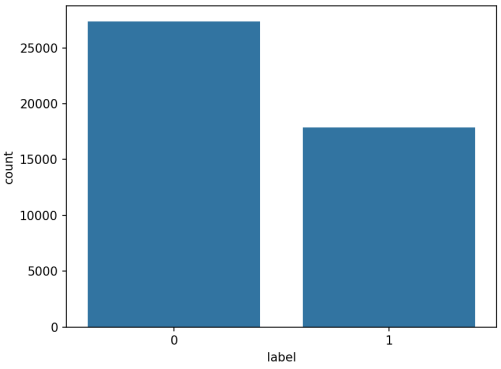


**Figure 2:** *Class distribution. Label* 0 *corresponds to human-written texts, while label* 1 *stands for AI-written texts.*

We then performed an in-depth analysis regarding word count, unique word count, and word per AI model. Here we just show some of our results.:

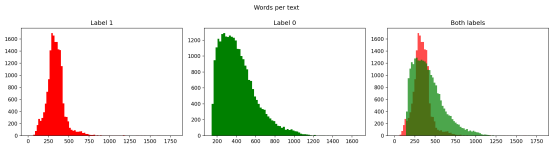


**Figure 4:** *Words per text per label.*

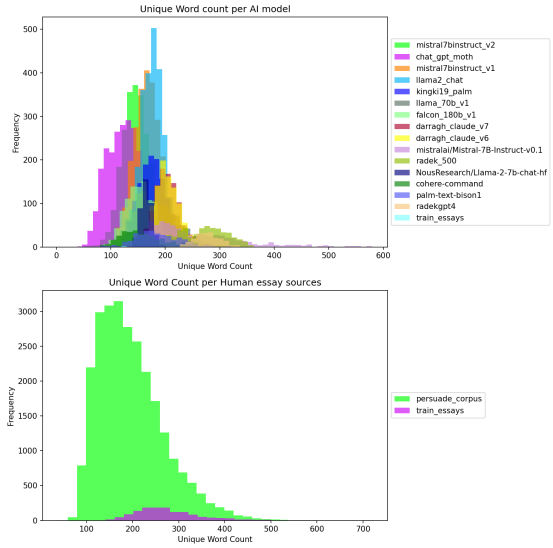


**Figure 5:** *The unique word count per essay.*

This analysis concluded that in general human-written essays use more words ($\sim$ 418) and more unique words per text ($\sim$ 200). Also, it is interesting how different certain AI models can be. (cf. Fig. 5 top figure.). This can serve as another basis for the tokenization or learning process, which can be investigated in a future project.

The full pre-processing pipeline can be found in our GitHub repo[2], in `scripts/preprocess_database_for_classifiers_ML.ipynb`.

## 4.2 Feature engineering

In this step, raw text data will be transformed into feature vectors and new features will be created using the existing dataset. We will implement the following different ideas to obtain relevant features from our dataset. This also serves as tokenization.

1. **Count Vectors as features:** Count Vector is a matrix notation of the dataset in which every row represents a document from the corpus, every column represents a term from the corpus, and every cell represents the frequency count of a particular term in a particular document.
2. **TF-IDF Vectors as features:** TF-IDF score represents the relative importance of a term in the document and the entire corpus. TF-IDF score is composed of two terms: the first computes the normalized Term Frequency (TF), and the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of documents in the corpus divided by the number of documents where the specific term appears.
    - $\text{TF(t)} = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}}$
    - $\text{IDF(t)} = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}}\right)$
    - TF-IDF Vectors can be generated at different levels of input tokens (words, characters, n-grams)

    (a) Word Level TF-IDF: Matrix representing tf-idf scores of every term in different documents
    (b) N-gram Level TF-IDF: N-grams are the combination of N terms together. This Matrix represents tf-idf scores of N-grams
    (c) Character Level TF-IDF: Matrix representing tf-idf scores of character-level n-grams
3. **Word Embeddings as features:** A word embedding is a form of representing words and documents using a dense vector representation. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. Word embeddings can be trained using the input corpus itself or can be generated using pre-trained word embeddings such as Glove, FastText, and Word2Vec. Any one of them can be downloaded and used as transfer learning.
4. **Text / NLP based features:** Several extra text-based features can also be created, which are sometimes helpful for improving text classification models. Some examples are:

    (a) Word Count of the documents – total number of words in the documents
    (b) Character Count of the documents – total number of characters in the documents
    (c) Average Word Density of the documents – average length of the words used in the documents
    (d) Punctuation Count in the Complete Essay – total number of punctuation marks in the documents
    (e) Upper Case Count in the Complete Essay – total number of uppercase words in the documents
    (f) Title Word Count in the Complete Essay – total number of proper case (title) words in the documents
    (g) Frequency distribution of Part of Speech Tags:
        - Noun Count
        - Verb Count
        - Adjective Count
        - Adverb Count
        - Pronoun Count

    These features are highly experimental and should be used according to the problem statement only.
5. **Topic Models as features:** Topic Modelling is a technique to identify the groups of words (called a topic) from a collection of documents that contains the best information in the collection. I have used Latent Dirichlet Allocation for generating Topic Modelling Features. LDA is an iterative model which starts from a fixed number of topics. Each topic is represented as a distribution over words, and each document is then represented as a distribution over topics. Although the tokens themselves are meaningless, the probability distributions over words provided by the topics provide a sense of the different ideas contained in the documents. We remark that this was a yet unsuccessful way to tokenize since we lack the knowledge even about the approximate number of topics from the essays.

## 4.3 Some well-known classifiers

Finally, we train the classifiers using the features created. There are many different choices of machine learning models that can be used to train a final model. We implemented a handful of them:

1. Naive Bayes Classifier
2. Linear Classifier
3. Support Vector Machine
4. Bagging Models
5. Boosting Models
6. Shallow Neural Networks
7. Deep Neural Networks
    - Convolutional Neural Network (CNN)
    - Long Short-Term Memory Model (LSTM)
    - Gated Recurrent Unit (GRU)
    - Bidirectional RNN
    - Recurrent Convolutional Neural Network (RCNN)
    - Other Variants of Deep Neural Networks

#### 4.3.1 Naive Bayes

Naive Bayes is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. See more here.

#### 4.3.2 Linear classifier

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic/sigmoid function. One can read more about logistic regression. See more here.

#### 4.3.3 SVM model

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification and regression challenges. The model extracts the best possible hyper-plane / line that segregates the two classes. See more here.

#### 4.3.4 Bagging model

Random Forest models are a type of ensemble model, particularly bagging models. They are part of the tree-based model family. One can read more about Bagging and random forests. See more here.

#### 4.3.5 Boosting models

Boosting models are another type of ensemble model part of tree-based models. Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones. A weak learner is defined to be a classifier that is only slightly correlated with the true classification (it can label examples better than random guessing). See more here.

### 4.4 Implementing our NNs as clasifiers

#### 4.4.1 Shallow Neural Network (SNN)

A neural network is a mathematical model that is designed to behave similarly to biological neurons and the nervous system. These models are used to recognize complex patterns and relationships that exist within labeled data. A shallow neural network contains mainly three types of layers – input layer, hidden layer, and output layer.

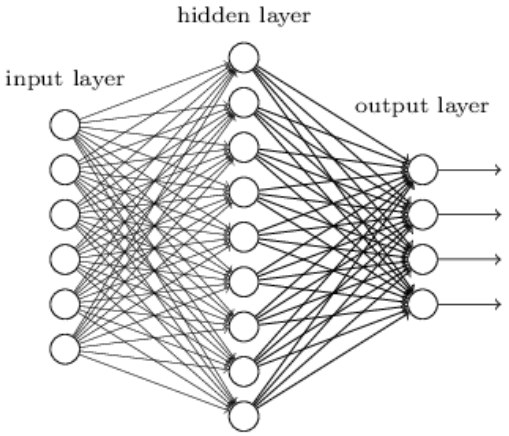


**Figure 6:** *The structure of aa SNN. It has only one hidden layer.*

#### 4.4.2 Convolutional Neural Network (CNN)

In a CNN, convolutions over the input layer are used to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters and combines their results.

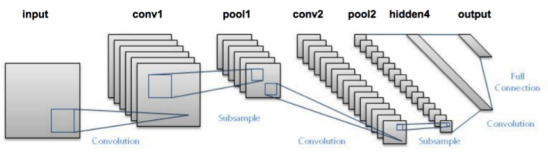


**Figure 7:** *Schematic structure of a CNN. Convolutions of the previous layers are used as inputs for the next layers.*

#### 4.4.3 Recurrent Neural Networks (RNN)

Unlike feed-forward neural networks in which activation outputs are propagated only in one direction, the activation outputs from neurons propagate in both directions (from inputs to outputs and from outputs to inputs) in Recurrent Neural Networks. This creates loops in the neural network architecture which acts as a 'memory state' of the neurons. This state allows the neurons an ability to remember what has been learned so far.

The memory state in RNNs gives an advantage over traditional neural networks but a problem called *vanishing gradient* is associated with them. In this problem, while learning with a large number of layers, it becomes really hard for the network to learn and tune the parameters of the earlier layers. To address this problem, a new type of RNNs called Long Short-Term Memory Models has been developed.

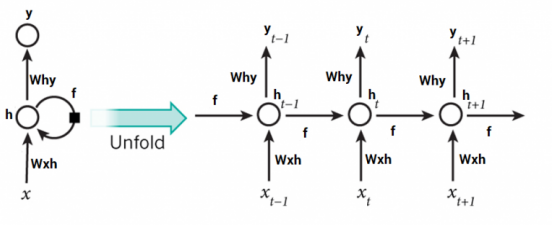


**Figure 8:** *Schematic structure of a typical RNN.*

Implemented RNNs:

1. Long Short-Term Memory (LSTM)
2. Gated Recurrent Units (GRU)
3. Bidirectional RNN (BRNN)
4. Recurrent Convolutional Neural Network (RCNN)

## 4.5 Results

We trained our implemented models with different tokenization combinations and test-training set combinations. The usual pattern was that we trained the models on cleaned texts (stemming, lemmatization, stopword removal, special character removal, etc. See our preprocessing notebook.) and tested both on cleaned and original texts. We repeated the same procedure taking original texts as the training set.

As for measure, we have used the area under receiver operating characteristic (area under ROC curve), as this is a widely accepted characteristic in such cases. We also plotted the confusion matrices for each model in each train-test case.

In the case of NNs, we used 20 epochs as training period, with usually 32 as batch size. The loss function was chosen for `'sparse_categorical_crossentropy'` and the activation function on the last layer was always `'softmax'` with 2 categories. This way, the output of the NNs was the same as that of the built-in classifiers.

In the case of the built-in classifiers, we used them on default parameters, except the SVM model, where we limited the number of iterations to 400 in case of original texts as training dataset and respectively to 1000 in the case of cleaned texts as training dataset. This was because we had to enable the prediction feature of the classifier which increased significantly the computation time and memory requirement.

The general conclusion is that NNs outperform simple classifiers when the testing and training sets are of the same type (i.e. they are both original or cleaned texts, see Fig. 9, 11, 10, 12), but when the test and training sets are different (original-cleaned or cleaned-original combination), the built-in, computationally cheap classifiers perform better (see Fig. 13, 15, 14, 16).

The exception among NN is surprisingly the SNN model, which is also an explanation, of why the others perform on the lower side. In case of the other NNs there are considerably more parameters as the number of hidden layers is larger compared to simple classifiers or SNN. Hence, the later layers learn slower the parameters of the first layers.

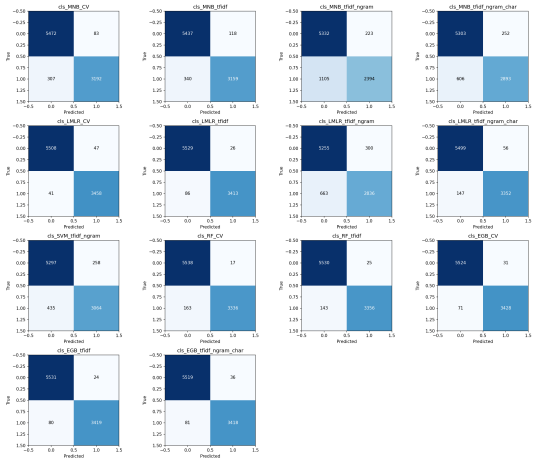

**Figure 9:** *Confusion matrix. Clean texts as training dataset, clean texts as testing dataset, built-in classifiers.*

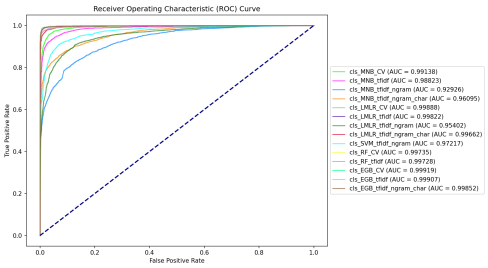


**Figure 10:** *ROC curve. Clean texts as training dataset, clean texts as testing dataset, built-in classifiers.*

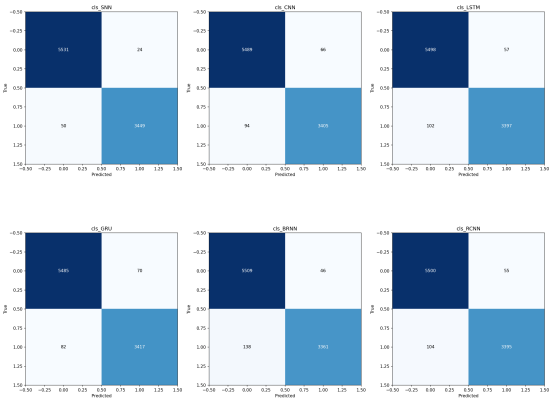

**Figure 11:** *Confusion matrix. Clean texts as training dataset, clean texts as testing dataset, implemented NNs.*

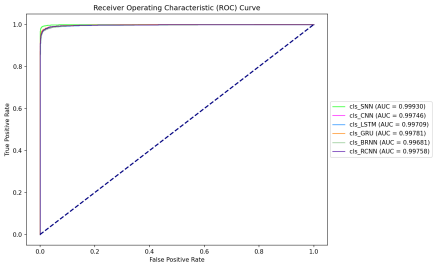


**Figure 12:** *ROC curve. Clean texts as training dataset, clean texts as testing dataset, implemented NNs.*

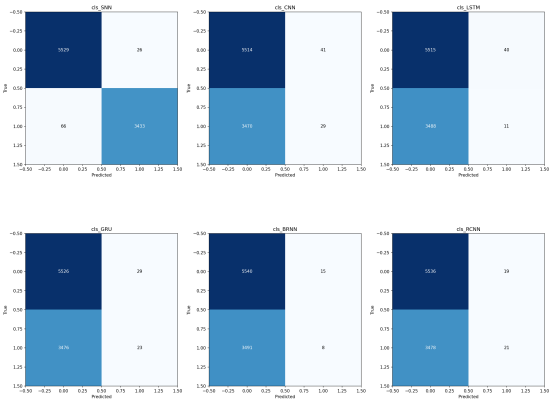


**Figure 15:** *Confusion matrix. Clean texts as training dataset, original texts as testing dataset, implemented NNs.*

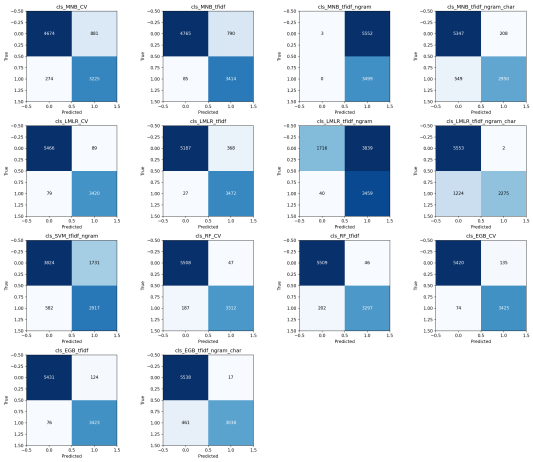


**Figure 13:** *Confusion matrix. Clean texts as training dataset, original texts as testing dataset, built-in classifiers.*

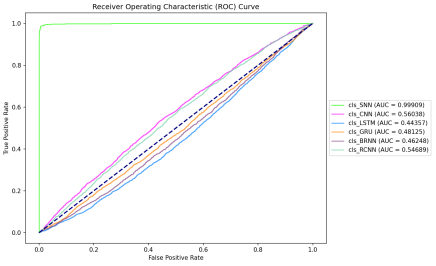


**Figure 16:** *ROC curve. Clean texts as training dataset, original texts as testing dataset, implemented NNs.*

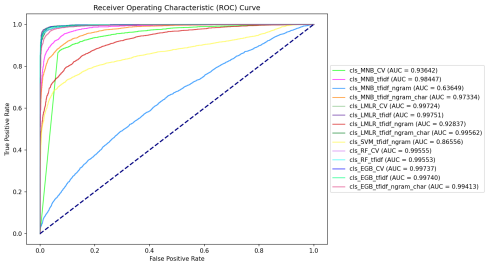


**Figure 14:** *ROC curve. Clean texts as training dataset, original texts as testing dataset, built-in classifiers.*

Other reasons among many why the NNs perform worse in these cases could be:

- **Data Bias:** Neural networks learn patterns from the data they are trained on. If the training data is biased towards a particular type of writing (e.g., mostly human-written text), the model may struggle to generalize well to the characteristics of AI-written text. If the model has not seen enough diverse examples of AI-generated text during training, it may not effectively learn to distinguish between human and AI writing.
- **Similarity in Language Patterns:** Advanced AI models, such as GPT-3, are designed to generate text that closely resembles human writing. These models have been trained on massive amounts of human-generated text and can mimic various writing styles. As a result, the language patterns in AI-generated text may be very similar to those in human-generated text, making it difficult for a classifier to identify the origin.
- **Lack of Specific Features:** Traditional classifiers might rely on specific features that distinguish between human and AI writing. Deep neural networks, especially those based on transformer architectures like GPT-3, work with distributed representations and may not have explicit features

that are easy to interpret or isolate for this specific task.

To overcome the lack of specific features and/or data bias we might have to later overview our tokenization and text cleaning processes.

The rest of our results and figures can be found in our GitHub repository[2] in the notebooks in the `scripts/` folder.

# 5 Text classification with ensemble model

## 5.1 Outline of the classification process

The next approach to human vs AI-written text classification was building an ensemble model of various classifiers. The idea behind this method is that aggregating in a given way the output of different, potentially diverse, and independent models yields better predictive power than each model individually. The aggregated classifier in our case was a soft voting classifier, where soft voting enables the ensemble model to predict the class label based on the maximum argumentum of the sums of the probabilities predicted by the smaller models.

For training the model, the above-mentioned `DAIGT_V2 dataset`[1] and our augmented dataset with additional 400 synthetic essays were used. 20% of the data was set aside for testing, the rest was used for hyperparameter tuning and training. Regarding the slight imbalance presented in the dataset, stratification on the label was used in the train-test-split method.

The preprocessing steps included tokenization and TF-IDF Vectorization. For the latter, we used scikit-learn `TfidfVectorizer` with already tokenized text which may be advisable when the tokenization process is complex. TF-IDF was chosen since it takes into account the frequently occurring words.

## 5.2 Classifiers of the ensemble model

The following classifiers were used in our ensemble model:

- **Random Forest:** This alone is an ensemble model too, using numerous decision trees. One of the most important parameters of this model is the number of estimators used.
- **Naive Bayes:** This classifier was mentioned above.
- **Logistic Regression:** We used a linear model, the logistic regression too, this classifier uses the log-loss function.
- **SGD classifier:** This is a regularized linear model, using stochastic gradient descent learning. Its loss function decides the nature of the linear model, here we use the `'modified-huber'`, which is a smooth function bringing tolerance to outliers and giving probability estimates.
- **Light Gradient Boosting Machine:** This boosting classifier uses decision trees that grow leaf-wise.

## 5.3 Hyperparameter tuning

Since the components of the ensemble model have numerous parameters that can affect the outcome of prediction, the performance of the ensemble model can potentially be enhanced by finding the optimal parameter values. Given that a reasonable parameter grid means excessive computational times, scikit-learn `RandomizedSearchCV` was chosen for the hyperparameter tuning.

Unfortunately, the hyperparameter tuning process was not finished due to time shortage and possible errors in the code, but here we list the considerations that were made while writing the relevant part of our code that can be found on Github[2].

The `RandomizedSearchCV` object was created with `'roc-auc'` scoring since the model performance is to be evaluated based on the AUC value of the ROC curve. We created a custom cross-validation splitter, that uses the scikit-learn `StratifiedKFold` splitter with stratification on the label variable. 3 fold was used. The parameter grid can contain parameters of the individual classifiers such as the number of estimators of the Random Forest.

We aimed to obtain the best parameters, update the ensemble model classifiers with them, and then train the model using all of the training data. Evaluation of the model performance was done using the test set.

## 5.4 Results

The performance of the ensemble model can be evaluated by obtaining the following metrics: we show the confusion matrix in Fig. 17. The ROC curve with the AUC value can be found in Fig. 18.

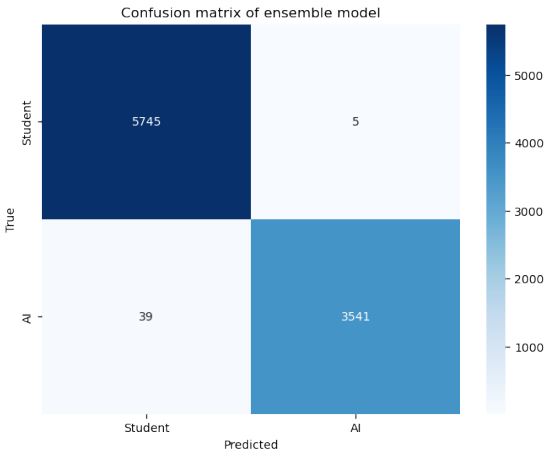


**Figure 17:** *Confusion matrix of the ensemble model. The Student label stands for the essays written by students, and the other is for the AI-generated texts.*

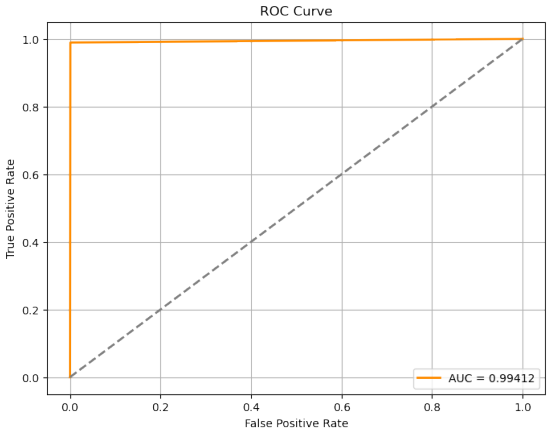


**Figure 18:** *ROC curve of ensemble model with AUC value.*

Compared to the results of the previous section (classification with different classifiers and neural networks), it can be seen that the ensemble model of simple classifiers performs in the range of the individual classifiers regarding the AUC metric. Depending on the tokenization method and text preprocessing, the ensemble model can potentially provide slightly better results than the individual classifiers. The neural networks (in the case of original texts) perform better than the ensemble model regarding the AUC values.

# 6 Conclusions

In our work, we have classified essays into two categories; AI and human-written ones. We wrote a pipeline for data augmentation by using the `Mistral-7B-Instruct-v0.1` LLM. We also experimented with other LLMs like Google's `gemini` or Meta's `Llama_2`.

We created several models, both based on built-in classifiers and by-hand-constructed NNs, and compared their performances in the case of different tokenizations and different train-test dataset pairs. We concluded that NNs perform better, when the training and testing sets' types are identic (both cleaned or original), while in other cases simple, computationally cheaper classifiers or SNNs are a better choice.

We have also created an ensemble model out of these classifiers and models and tried performing hyperparameter tuning. Results show that the ensemble model performance is similar to the built-in classifiers', showing promise to increase predictive power when the same tokenization and preprocessing procedure is used. The NNs give better results compared to the ensemble model of the built-in classifiers (using original texts as training and testing sets). The hyperparameter tuning could further elevate the results of the ensemble model, our plans include working on this part of the project.

# Bibliography

[1] Darek Kleczek. *DAIGT V2 Train Dataset*. 2023. URL: https://www.kaggle.com/datasets/thedrcat/daigt-v2-train-dataset.

[2] Marcell Emmer et al. *Detect AI-Generated Text*. Version 2.0.4. Jan. 2024. URL: https://github.com/emmermarcell/Detect-AI-Generated-Text.

[3] Zulqarnain Ali. *Explained LLM Model*. 2023. URL: https://www.kaggle.com/code/zulqarnainali/explained-llm-model.

[4] Abubakar Abid et al. *NLP Course, Hugging Face*. URL: https://huggingface.co/learn/nlp-course/chapter0/1?fw=tf.

[5] Shivam Bansal. *A Comprehensive Guide to Understand and Implement Text Classification in Python*. 2022. URL: https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/.

[6] Vijaya Rani. *NLP Tutorial for Text Classification in Python*. 2021. URL: https://medium.com/analytics-vidhya/nlp-tutorial-for-text-classification-in-python-8f19cd17b49e.

[7] Saravana Kumar. *Machine Learning Model for Distinguishing Human vs. ChatGPT Text*. 2023. URL: https://github.com/saro0307/AI-detector.

[8] Wikipedia. *Receiver operating characteristic*. URL: https://en.wikipedia.org/wiki/Receiver_operating_characteristic.