AI-Human Text Detector

Haim Goldfisher   Itamar Kratiman

May 6, 2024

Project Repository: `https://github.com/haimgoldfisher/Ai-Human-Generated`

Abstract

This study introduces an AI-human text Detector designed to distinguish between human-written and AI-generated text, emphasizing interpretability and practicality. Utilizing a dataset sourced from Kaggle [1] comprising over 487,000 text samples, we address data imbalances, conduct extensive feature engineering, and explore various classification models. Employing the Doc2Vec algorithm for text vectorization, we enhance semantic representation using fixed-size vectors. Our analysis includes a deep dive into feature importance, model performance evaluation, and the creation of a user-friendly graphical interface (GUI) for real-world implementation. Notably, the Support Vector Machine (SVM) exhibits 99% accuracy, demonstrating robust performance in text classification tasks. This research contributes to the development of effective tools for combating text-based misinformation and maintaining academic integrity in an era dominated by AI-generated content.

1 Introduction

Since November 2022, when OpenAI introduced ChatGPT 3, the world of LLMs has been spreading like wildfire. LLMs have been introduced almost every month, and ease the ability to create text, based on simple instructions. Those LLM models are helpful in a large number of life aspects. They give the ability to expand knowledge easily, write essays faster, and more. Those LLMs also became a useful tool for students, to overcome difficulties in their studies. This phenomenon has spread so much that students have started cheating and handing in assignments fully generated by AI. It forces lectures to develop abilities to identify whether the students "cheated" or wrote the assignment themselves.

1.1 The Challenge

In this project, we managed to build a model that could help the ones standing on the side, from paper and website readers to academic lectures, to identify whether a human being wrote the text they read, or was penned by an AI tool. This makes our problem a binary classification problem, where the goal is to categorize the text into one of two classes: human-written or AI-generated. We also aim to grasp how confident the model is in its decision-making process. This is essential to prevent scenarios where the model might have slight doubts about whether the text was authored by artificial intelligence, potentially misattributing it to a human writer.

1.2 Authors and Human-AI Similarity

In addition to our primary objective of distinguishing between human-written and AI-generated text, we explore the project from the perspective of binary classification. This perspective categorizes the article writer as either human or AI, leveraging insights from research such as [2]. The study referenced investigates converting texts into POS (Part-of-Speech) n-grams to examine text patterns for categorization by an article writer. While this approach presents intriguing possibilities, we identify several potential challenges:

1.2.1. **Diverse Writing Styles:** Both AI and humans exhibit diverse writing styles, leading to variability within each category. Some humans may write in a manner resembling AI, and vice versa, complicating the classification task.

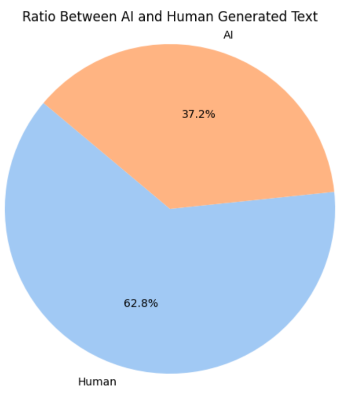
1.2.2. **Limited Model Training Data:** The model discussed in the referenced study may have only encountered a limited number of human and AI writing styles during training. Without exposure to a wide range of writing styles, the model's performance may suffer when confronted with unfamiliar texts.

By acknowledging these challenges, we underscore the importance of comprehensive model training and the need for robust evaluation methods to ensure accurate classification results.

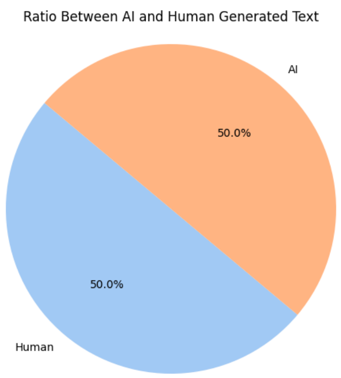
2 Dataset - Reading, Balancing, and Cleaning

Our dataset was taken from Kaggle. The dataset contains 487,235 text samples, divided into two classes- 0 for Human-generated text, and 1 for AI-generated text. The dataset suffers from a significant imbalance.

2.1. **Data Cleaning and Undersampling:** Our first attempt was to add samples of the minority class, which caused memory and runtime issues, so we decided to choose 10,000 samples from each class randomly. Before doing it, we cleaned the data, by making sure that there were no NaN and duplicated values and removing all escape sequences, such as '\n', '\t' etc. Moreover, texts that were too short were dropped (texts which contained less than 5 words).



(a) Before Undersampling



(b) After Undersampling

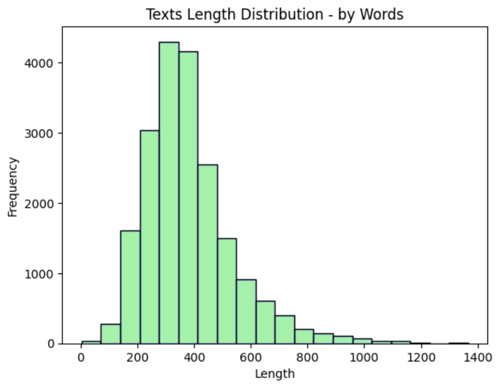
Figure 1. Ratio between Classes

2.2. **Text length:** The general text length is distributed as follows: The minimal length is 5 words only, the maximal length is 1,353 words and the average is 377. We can also look at the distribution of the lengths of each class and see that there is not a big difference between the classes.

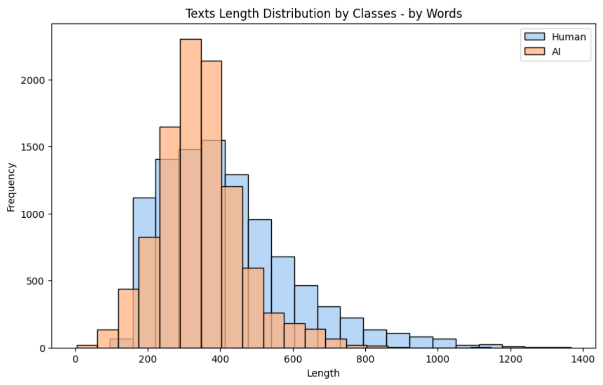
2.3. **Tokenization:** In natural language processing, tokenization emerges as an essential initial step in text preprocessing. Its purpose is to separate the words from a string in the correct way. This entails discarding punctuation marks, standardizing upper cases, consolidating words with different inflections, and removing stop words to streamline text complexity. By doing so, the ultimate goal is to facilitate model comprehension by recognizing identical words or those with similar semantic meanings. Acknowledging that determining the text author has a lot of dependence on the way the text was written as much as its content, one of the core ideas of our project was to preserve as much information as possible from the text, without harming the tokenization process. That is, we tokenized, but we wanted to preserve pattern information and not just semantic information. We did this while converting subtle details in the indices that we will talk about later.

3 Previous Attempts

Our first thought was to apply some NLP techniques and algorithms, and finally rejected them all, each for a specific reason.



(a) Full Data



(b) Divided into Classes

Figure 2. Text Length Distribution

3.1. **BoW - Bag of Words:** Bag of words is a NLP technique, that counts the number of words in the text. The main drawback is that it happened without considering the order of the words in the text. As a result, texts such as "I love my children" and "My children love me" are considered identical, although they have completely different meanings. In addition, large-scale vocabulary requires very long vectors, which is a phenomenon we would like to avoid.

3.2. **N-Grams:** N-Grams is a fantastic approach to examining the text's structure, because of its ability to consider N words as one unit. This technique is effective mostly when taking a few values of N and not only one. The main disadvantage that brought us to the decision to reject this technique, is the fact that the data set is heavy and contains a large amount of samples, therefore the process is inefficient in terms of runtime and space.

3.3. **Word2Vec:** Word2Vec is NLP technique, that translates each word to a vector. The idea behind it is that words with similar semantic meaning, are represented by similar vectors. The problem with this approach is not the technique itself, but the reason that it is not suitable to our objective, because it is too local. We are interested in a meaning of a text, and not of separated words.

3.4. **TF-IDS vectorization:** TF-IDF is a common NLP technique in text classification. It converts the text into a vector where each word receives an importance inside the text. However, it costs a very large space which our local machines could not provide.

4 EDA - Exploratory Data Analysis

To optimize the EDA process, we combined it with the feature engineering process. Therefore, we divided the process into two types: Simple: Features with easy creation, without external modulus. Complex: Features that require external modulus, such as NLTK.

4.1 Simple Features

4.1.1. **Punctuation:** We assume that in general, people may not use punctuations correctly, while AI is trained to do so, so it would be an effective feature to investigate. We calculated the total frequency of punctuations in each class and computed the percentage of punctuations in each sample. Since every text has a different length, the idea was to calculate the ratio between the punctuation and the number of words in the text.

4.1.2. **Richness:** The measure of richness is the number of unique words in the text. We used it but truly think this is not a reliable measure because of the text length distribution.

4.1.3. **Stop Words:** Correct spelling has rules for using stop words. Therefore we assume that the number of occurrences of stop words can be a difference between AI-generated texts and Human-generated texts, for the reason that humans might not know how to use them correctly, we are indifferent to it.

4.1.4. **Upper Case:** Like stop words, using upper cases has rules, so again, we assume that humans might use them incorrectly.

4.1.5. **Comma, Period, Question Mark:** We assume that there would be a difference between AI-generated texts and Human-generated text in the use of commas, periods, and question marks, as a result, that humans might use that punctuation in a wrong way. For example, people sometimes are likely to use a period instead of a comma.

4.1.6. **Average Sentence Length:** Correct English, would be written with short sentences. We assume that humans are less careful in this regard than AI. We can see that clearly in Fig 3a.

4.1.7. **Text Length:** Described above in the Dataset section

4.1.8. **Flesch Reading Ease:** This metric calculates how easy a piece of text is to understand. The higher the score, the easier it is to read. Scores can range from 0 to 100, where higher scores indicate easier readability. Lower scores indicate harder readability. Typically, scores above 60 are considered easily understandable, while scores below 30 indicate very difficult text. The formula for the Flesch Reading Ease score is based on the average sentence length and the average number of syllables per word. AI-generated texts.

4.1.9. **Flesch-Kincaid Grade Level:** This metric is similar to the Flesch Reading Ease score but instead of giving a score, it estimates the grade level needed to understand the text. For example, a score of 8.0 means that an eighth-grader would be able to understand the text. It is calculated using the average number of words per sentence and the average number of syllables per word.

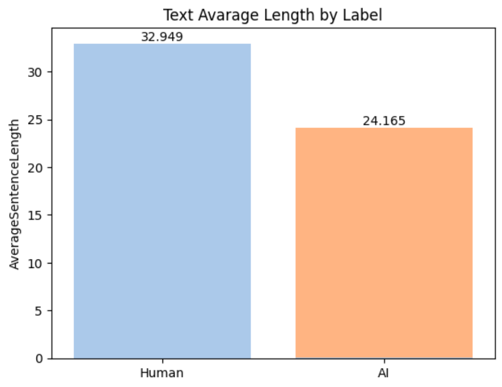
4.1.10. **Gunning Fog Index:** This index estimates the years of formal education a person needs to understand the text on the first reading. It is calculated using the average sentence length and the percentage of complex words (words with three or more syllables).

4.1.11. **Syntactic Complexity:** This metric measures the complexity of sentence structures in the text, according ratio between independent and subordinate clauses. An independent clause, also known as a main clause, is a group of words that can stand alone as a complete sentence. It expresses a complete thought and contains both a subject and a predicate. A subordinate clause, also called a dependent clause, is a group of words that cannot stand alone as a complete sentence because it does not express a complete thought. Instead, it depends on an independent clause to form a complete sentence. We assumed that Human-generated texts would be more complex than AI-generated texts, for the reason that humans might tend to use words that do not fit the sentence or use them in the wrong syntax. We can see it clearly in Fig 3b.

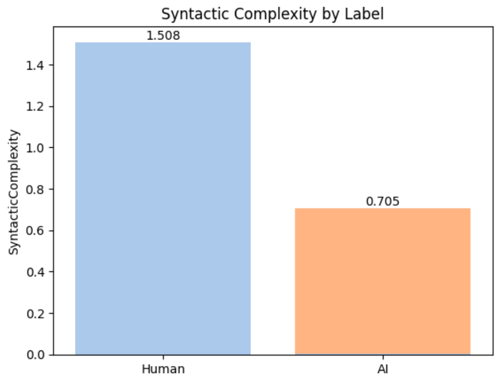
4.2 Complex Features

4.2.1. **Sentiment (neg, neu, pos, and compound):** The Sentiment feature, is divided into 4 separate features, each indicating the level of different sentiment in the text, 'neg' for negative, 'neu' for neutral, 'pos' for positive, and 'compound' which is a normalized compound score that combines all the other sentiments into an overall sentiment score. Those four features were built by the SentimentIntensity-Analyzer class of the NLTK module, which employs a lexicon-based approach to sentiment analysis. We can see that AI texts have a higher average compound score in Fig 4a.

4.2.2. **Part of Speach Tags:** There are eight parts of speech in the English language: noun, pronoun, verb, adjective, adverb, preposition, conjunction, and interjection.



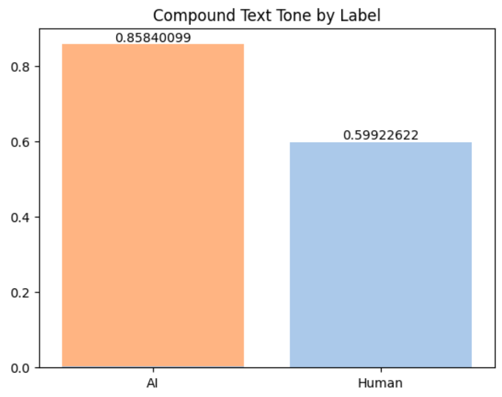
(a) Average Sentence Length Comparison



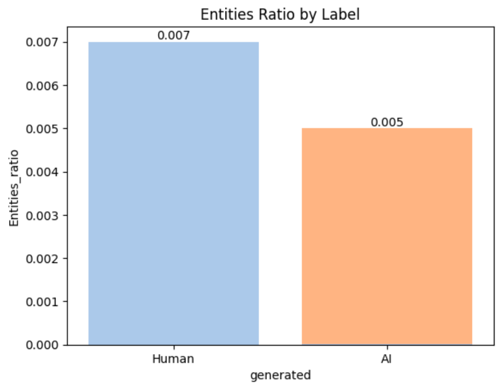
(b) Syntactic Complexity Comparison

Figure 3. Some Simple Features Comparisons

4.2.3. **Entities Ratio:** Entities, in the context of natural language processing (NLP), are specific pieces of information within a text that represent real-world objects such as persons, organizations, locations, dates, quantities, etc. From the following graph, we can see that the average entity ratio in both classes is negligible in Fig 4b.



(a) Sentiment Intensity Comparison



(b) Entities Ratio Comparison

Figure 4. Some Complex Features Comparisons

5 Train-Validation-Test Split

We partition our dataset into three subsets: training, validation, and test sets. Initially, we divide the entire dataset into training and the rest using an 80-20 split. Then, we further split the remaining data equally into validation and test sets, each comprising 50% of the remaining dataset. During this process, we ensure to remove the "generated" column from all subsets to separate the input features from the labels. This systematic splitting strategy ensures that we have distinct datasets for training, validation, and testing, enabling robust evaluation of our models' performance.

6 Standardization

After extracting the features from the textual data, we apply standardization using the StandardScaler from Scikit-Learn. Standardization rescales the features to have a mean of 0 and a standard deviation of 1, ensuring that each feature contributes equally to the analysis. This process is essential for models that rely on distance-based algorithms or regularization techniques, as it prevents features with larger scales from dominating the model (e.g., "Average Sentence Length" values are always greater than 1). The StandardScaler is used to transform each feature independently, preserving the shape of the distribution while centering it around the mean. Furthermore, it's important to note that the standardization process is performed solely on the training set. Once the scaler is fit to the training data and the features are transformed, the same scaler is then applied to the validation and test sets.

7 Regularization

We employ L1 regularization with linear regression solely to eliminate unnecessary columns from the dataset. L1 regularization, also known as Lasso, introduces a penalty term proportional to the absolute value of the coefficients. By doing so, it encourages sparse solutions, effectively driving some coefficients to zero. In our case, applying L1 regularization helps to automatically select relevant features while discarding irrelevant ones. Features with coefficients close to zero or very small impact on the target variable include "neu", "% uppercase", and "Interjections" which had dropped due to L1. Features like "AVG sentence length", "Flesch Reading Ease", "Flesch Kincaid Grade Level", "Gunning Fog Index", and "Syntactic Complexity" had relatively high coefficients, indicating their importance in the model.

8 Vectorization with Doc2Vec

In our study, we leverage the *Doc2Vec* algorithm, an extension of the popular Word2Vec model, to convert our text data into fixed-size vectors. Developed as part of the Gensim library, Doc2Vec extends the capabilities of Word2Vec by incorporating document-level context alongside word-level context. This approach is inspired by the work of Tomas Mikolov and his team at Google, particularly the paper titled "Distributed Representations of Sentences and Documents" [3] published in 2014.

8.1 Understanding Doc2Vec

Doc2Vec operates on the same principles as Word2Vec but extends them to entire documents rather than individual words. Like Word2Vec, Doc2Vec uses neural networks to learn distributed representations of words (Word Embeddings) or documents (Document Embeddings) in a continuous vector space. It offers two main algorithms for training: PV-DM (Paragraph Vector - Distributed Memory) and PV-DBOW (Paragraph Vector - Distributed Bag of Words). PV-DM aims to predict the next word in a context window given both the context words and a unique document vector. PV-DBOW, on the other hand, predicts words in a window using only the document vector, disregarding the context words. To train the Doc2Vec model, we exclusively use the training dataset. During training, the model learns to associate each document with a unique vector representation, capturing the semantic meaning of the text. Once trained, the Doc2Vec model can generate document vectors for text in the training, validation, and test datasets using the *infer_vector* method. This method infers the document vector based on the learned parameters of the model without updating them. The generated document vectors serve as fixed-size representations of the original text, facilitating downstream machine-learning tasks. The generated document vectors replace the original "text" column in our datasets, reducing dimensionality while preserving semantic information. Each vector typically consists of 50 dimensions, representing a compact yet informative representation of the text.

9 Modeling - Overview

In this section, we explore various classification algorithms to train predictive models on our dataset. Here's an overview of the process:

9.1. **Model Selection:** We consider a range of classification models, including Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), AdaBoost, Decision Tree, and Random Forest.

9.2. **Hyperparameter Tuning:** Each model undergoes hyperparameter tuning using grid search cross-validation (GridSearchCV). Hyperparameters and their corresponding parameter grids are predefined for optimization.

9.3. **Classification Pipeline:** We define a classification_pipeline function to streamline model evaluation. This function performs grid search cross-validation to find the best hyperparameters and evaluates model performance on the validation set.

9.4. **Evaluation Metrics:** Standard classification metrics such as precision, recall, F1-score, and area under the ROC curve (AUC) are utilized. These metrics provide insights into predictive performance and class differentiation ability.

9.5. **Results Visualization:** Classification reports, confusion matrices, and ROC curves are visualized for each model to interpret performance. The classification report offers detailed class metrics, while the confusion matrix illustrates prediction outcomes. The ROC curve depicts the true positive rate vs. false positive rate trade-off across thresholds.

10 Modeling - Models and Scores

In this section, we trained various Machine Learning algorithms using grid search to find the best parameter for the specific problem (fig 11). We evaluated them using typical classification metrics, such as Precision, Recall, and F1-score, using a confusion matrix, alongside with ROC curve to understand the TPR-FPR rate. We also took into account the rate between misclassified samples. We wanted the number of AI-generated samples which misclassified as human-generated, to be significantly higher than the opposite case, where human-generated texts were classified as AI. This thought and mindset are considered in detail in the Results section below.

Table 1. Model Evaluation Metrics

Model	**Precision**	**Recall**	**F1-Score**	**AUC-ROC**	**Misclassified Samples**
Logistic Regression	0.97	0.97	0.97	0.99	65 (41, 24)
KNN	0.96	0.96	0.96	0.99	72 (25, 47)
SVM	0.99	0.99	0.99	1.00	27 (20, 7)
AdaBoost	0.96	0.96	0.96	0.99	73 (38, 35)
Decision Tree	0.90	0.90	0.90	0.95	208 (106, 102)
Random Forest	0.94	0.94	0.94	0.99	121 (87, 34)

11 Results

According to the previous section, we can observe that the SVM model has the best performances and scores, for all metrics- Precision (0.99), Recall (0.99), and F1-Score (0.99) alongside with an accuracy of 0.99.

11.1 Testing Optimal Model with Test Set

One of the most important actions in ML is to test the optimal model on a test set, which it has never seen. This action is performed during the validating process using the validation set, but also at the end of the process using the test set. We performed this action on the SVM model and received a Precision, Recall, and F1-Score of 0.99. Those excellent results indicate a well-trained model.

11.2 Threshold Adjusment

Imagine a scenario when a great student is handed an assignment, and the lecturer uses an AI-Human-Classifier which classifies it as an AI-generated text. This mistake would create an unnecessary, and unjust

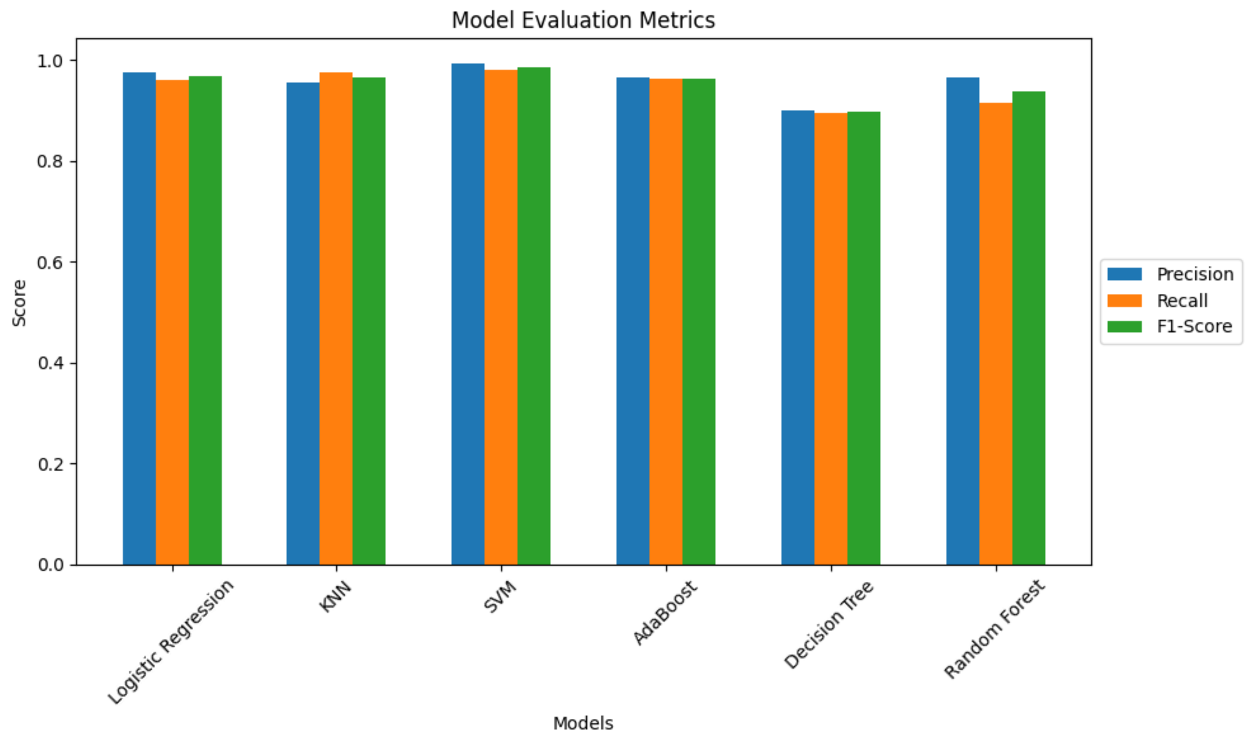


Figure 5. Models Results Summary.

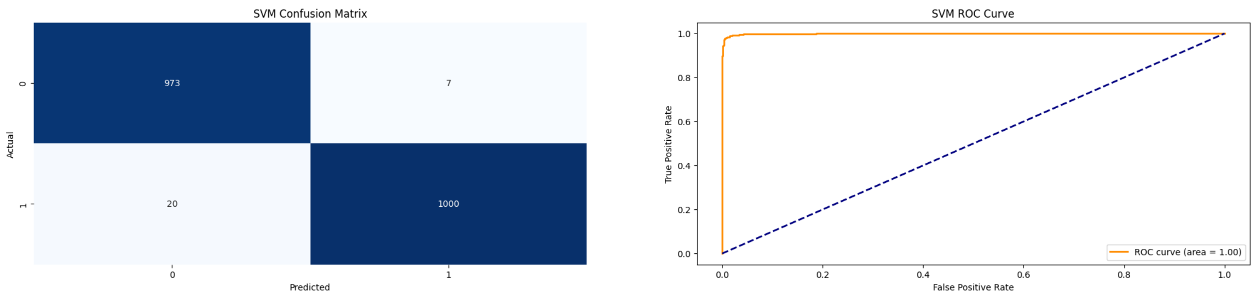
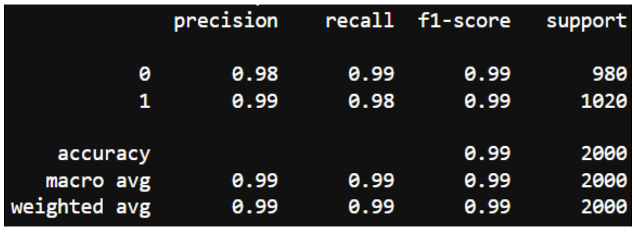


Figure 6. SVM Confusion Matrix and ROC Curve

situation. To prevent scenarios like this, we applied a threshold to our classifier, forcing it to receive a probability of at least 0.81 to classify a text as AI-generated. This threshold may be chosen by the user.

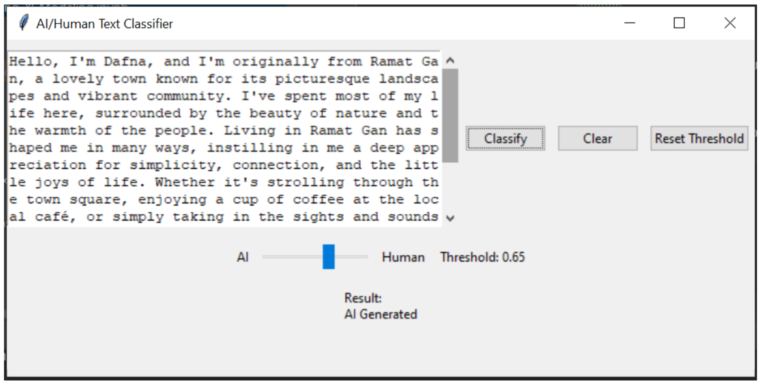


```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.98      0.99      1020

    accuracy                           0.99      2000
   macro avg       0.99      0.99      0.99      2000
weighted avg       0.99      0.99      0.99      2000
```

Figure 7. SVM Classification Report on Test with 0.81 Threshold.

11.3 External Validation

To conduct external validation and test the model in real-world scenarios, we exported the model as a pickle (pkl.) file and integrated it into a graphical user interface (GUI). We also brought the option to control the threshold. After some self-tests, we have noticed that the model is struggling in short texts. This phenomenon surprised us since our model has nearly 100% of success at test data. It made us rethink the distribution of the dataset and the variety of data sources. We will discuss it in the next section.



```
Hello, I'm Dafna, and I'm originally from Ramat Ga
n, a lovely town known for its picturesque landsca
pes and vibrant community. I've spent most of my l
ife here, surrounded by the beauty of nature and t
he warmth of the people. Living in Ramat Gan has s
haped me in many ways, instilling in me a deep app
reciation for simplicity, connection, and the litt
le joys of life. Whether it's strolling through th
e town square, enjoying a cup of coffee at the loc
al café, or simply taking in the sights and sounds
```

Threshold: 0.65

Result:
AI Generated

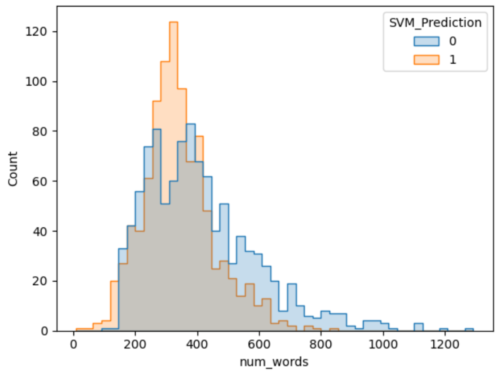
Figure 8. GUI Example

11.4 Model Results Analysis

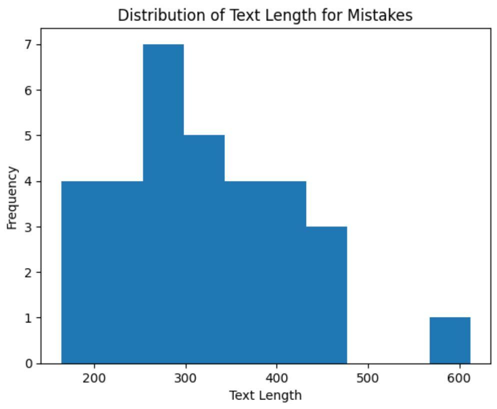
To gain insights into the strengths and weaknesses of our model, we conducted an analysis based on text length. Our hypothesis points out that the model's performance might vary across different text lengths, with potentially stronger performance on texts closer to the average data length of approximately 377 words. After evaluating our best model (SVM), we found a total of 27 mistakes. We then analyzed the distribution of these mistakes across the different text lengths (see Fig 9). We can infer from it that there is no correlation between the length of the text and the performance of the model (see Fig 9b). However, it's noteworthy that the shortest texts exclusively belong to the 'AI' class, while the longest texts are solely from the 'Human' class (see Fig. 9a). This distributional characteristic could potentially impact the model's performance on real-world tests.

11.5 Data Split Test

In order to assess the effectiveness of our approach, we conducted a data split test to evaluate the performance of our models using two distinct feature sets: doc2vec features and feature engineering features. We selected three significant models for this analysis: Support Vector Machine (SVM), K-Nearest Neighbors (KNN),



(a) SVM Predictions



(b) SVM Miscalculations

Figure 9. SVM Predictions in Terms of Text Length

and Decision Tree. After splitting the data into these two groups, we trained each model individually to investigate the impact of each feature set on model performance. The results of our analysis yielded insightful observations.

Model	Doc2Vec	Feature Engineering	United Data
SVM	97.5%	94%	99%
KNN	94.25%	90.45%	96%
Decision Tree	81.5%	85.85%	90%

Table 2. Summary of Model Performance with Different Feature Sets

As depicted in Table 2, the performance varied across different models and feature sets. In SVM and KNN models, the utilization of doc2vec features yielded superior results compared to feature engineering features. Conversely, in the Decision Tree model, the feature engineering approach outperformed the doc2vec features.

12 Conclusions & Future Directions

12.1. Since our best approach involved vectorizing the text using a neural network-based algorithm, SVM emerged as the best model. SVM's ability to handle high-dimensional data, find optimal separating hyperplanes, and perform well in situations with a large number of features make it a suitable choice for text classification tasks. Notably, most of the ML models we utilized performed well, where linear separation-based algorithms were the best. This suggests that the most significant task lies in the vectorization process. Doc2vec stands out as the most impactful model in our approach, as it simplifies the problem, making the job of the classifier much easier after this process.

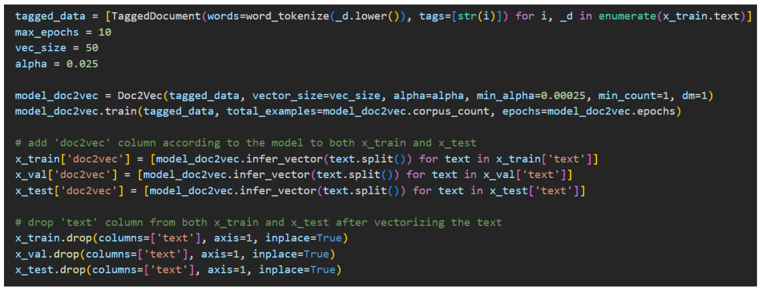
12.2. A notable aspect that required further consideration was why KNN outperformed decision tree-based algorithms (Decision Tree, and Random Forest). Despite KNN being a local method that lacks a "learning process," it exhibited superior performance compared to decision tree methods. We suggest that there is a clear correlation among the features themselves, so if the text is generated by a human, there should be a clear trend in most features. Sufficiently close vectors representing the text and also created by a human must be close enough for KNN to successfully find similar texts/vectors also created by humans. There must be such a correlation, otherwise, such cells and others within the vector would cancel each other out, and then most vectors would be mixed in terms of their group (the metric is distance, so if features are likely to cancel each other out, in an approach like KNN that lacks any learning process or weighting of features, the model would perform poorly). Conversely, decision tree methods may have struggled due to insufficient depth in capturing relevant features from the data (in grid search we defined 'max depth' to be the best from 3,5,7). This disparity underscores the importance of feature selection in achieving optimal performance in classification tasks.

12.3. As mentioned earlier, in this project, we randomly undersampled the data. A more effective approach would involve equal sampling to preserve the distribution of the data. Additionally, we could improve by sampling more from weaker texts, particularly shorter ones. Moreover, our dataset was sourced from a single dataset without detailed information about its origin. We cannot consider the test data as a true representation of error, as it may not encompass all texts worldwide.

12.4. With more powerful computing resources, we could utilize more data and build stronger models. The ideal model would draw data from various sources to train on a plethora of AI-generated texts. Moreover, Doc2vec vectors could be longer. We opted for 50-sized vectors to conserve memory space, given our limitations in CPU strength.

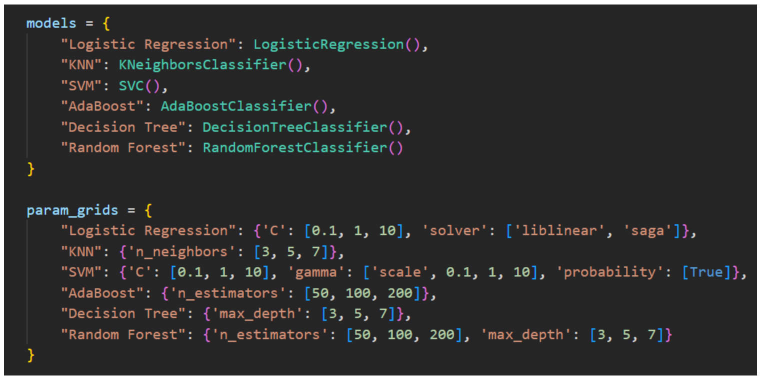
12.5. As technology continues to advance, the capability to create smarter AI grows. A proficient text detector must undergo continuous training with new data to learn emerging AI techniques.

13 Code Snippets



```python
tagged_data = [TaggedDocument(words=word_tokenize(_d.lower()), tags=[str(i)]) for i, _d in enumerate(x_train.text)]
max_epochs = 10
vec_size = 50
alpha = 0.025

model_doc2vec = Doc2Vec(tagged_data, vector_size=vec_size, alpha=alpha, min_alpha=0.00025, min_count=1, dm=1)
model_doc2vec.train(tagged_data, total_examples=model_doc2vec.corpus_count, epochs=model_doc2vec.epochs)

# add 'doc2vec' column according to the model to both x_train and x_test
x_train['doc2vec'] = [model_doc2vec.infer_vector(text.split()) for text in x_train['text']]
x_val['doc2vec'] = [model_doc2vec.infer_vector(text.split()) for text in x_val['text']]
x_test['doc2vec'] = [model_doc2vec.infer_vector(text.split()) for text in x_test['text']]

# drop 'text' column from both x_train and x_test after vectorizing the text
x_train.drop(columns=['text'], axis=1, inplace=True)
x_val.drop(columns=['text'], axis=1, inplace=True)
x_test.drop(columns=['text'], axis=1, inplace=True)
```

Figure 10. Text Embedding with Doc2Vec: Vectorization & Feature Engineering



```python
models = {
    "Logistic Regression": LogisticRegression(),
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(),
    "AdaBoost": AdaBoostClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier()
}

param_grids = {
    "Logistic Regression": {'C': [0.1, 1, 10], 'solver': ['liblinear', 'saga']},
    "KNN": {'n_neighbors': [3, 5, 7]},
    "SVM": {'C': [0.1, 1, 10], 'gamma': ['scale', 0.1, 1, 10], 'probability': [True]},
    "AdaBoost": {'n_estimators': [50, 100, 200]},
    "Decision Tree": {'max_depth': [3, 5, 7]},
    "Random Forest": {'n_estimators': [50, 100, 200], 'max_depth': [3, 5, 7]}
}
```

Figure 11. Classification Models and Hyperparameter Grids

References

[1] *AI Vs Human Text Dataset*: `https://www.kaggle.com/datasets/shanegerami/ai-vs-human-text`

[2] Rahul Radhakrishnan Iyer and Carolyn Penstein Rose. *A Machine Learning Framework for Authorship Identification From Texts*: `https://doi.org/10.48550/arXiv.1912.10204`

[3] Quoc V. Le and Tomas Mikolov. *Distributed Representations of Sentences and Documents*: `https://doi.org/10.48550/arXiv.1405.4053`