IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

Classification Of Human Written Text And Artificial Intelligence Generated Text Using Machine Learning

Azmat Rashid[1], Dr. Gurinder Kaur Sodhi [2]

[1]M. Tech Scholar, Department of Electronics and Communications Engineering, Desh Bhagat University, Punjab, India
[2] Assistant Professor, Department of Electronics and Communications, Desh Bhagat University, Punjab, India

Abstract: This study explores how machine learning techniques can classify text data as either human-written or AI-generated. The research uses a dataset containing 6,000 texts, evenly split between human and AI-generated samples. Key objectives include preprocessing the data, conducting exploratory data analysis (EDA), training models, and thorough evaluation. Text data underwent detailed preprocessing steps such as tokenization, removal of stop words and punctuation, and lemmatization. EDA revealed insights into distributions of text lengths and labels, highlighting significant differences between human and AI-generated texts. Two models, Naive Bayes and Logistic Regression, were trained on the processed dataset. Logistic Regression outperformed with 98% accuracy on a separate test set. Metrics for AI-generated texts showed precision, recall, and F1-score at 98%, 95%, and 97%, respectively, showcasing its strong ability to distinguish between human and AI-generated content. The results emphasize the effectiveness of advanced text preprocessing and machine learning models in accurately identifying text origins. Logistic Regression emerged as the preferred model due to its high accuracy and comprehensive performance metrics, making it a dependable tool for distinguishing between human and AI-generated content across various applications.

Keywords: Written text, text generation, Machine learning

I. INTRODUCTION

The rise of artificial intelligence (AI) in digital communication has transformed how content is generated, especially with its ability to produce text that closely mimics human writing. This capability, driven by advancements in natural language processing (NLP), has revolutionized content creation across online platforms. AI-generated text is now pervasive, blurring the distinction between human and machine-authored content, from automated news articles and reviews to personalized chatbot responses and social media posts.

While AI-driven content creation promises efficiency and personalization benefits, it also presents significant challenges. One major concern is the misuse of AI to spread misinformation, manipulate public opinion, and undermine the credibility of online information sources. Unlike traditional content, AI-generated text can be crafted with remarkable sophistication, making it difficult for users and platforms to verify its authenticity. Therefore, accurately identifying AI-generated text has become crucial to safeguarding the integrity of digital content ecosystems. This thesis tackles this critical issue by exploring advanced machine learning techniques and computational methods to develop robust detection models. These models aim to distinguish between human and AI-generated text effectively, enabling platforms to mitigate the risks associated with deceptive or misleading content.

In recent years, advancements in NLP have empowered AI models to produce coherent and contextually relevant text across various domains, including news, social media, and customer reviews. While these capabilities are groundbreaking, they have also raised concerns about the credibility and trustworthiness of digital content. Differentiating between human-generated and AI-generated content is now essential due to the potential for misinformation and the challenges posed by the sheer volume of content online. Traditional methods of content verification struggle to keep up with the scale and speed of AI-generated content dissemination. Thus, there is an urgent need for automated tools that can accurately detect and flag AI-generated text in real-time.

Technologically, addressing this challenge involves leveraging advances in machine learning, NLP, and computational linguistics. Techniques such as supervised learning, deep learning, and ensemble methods are employed to develop robust AI detection models. These models are trained on large datasets encompassing both human and AI-generated text to learn distinctive features that differentiate between the two. Moreover, ethical considerations surrounding the use of AI detection technologies are carefully considered. Effective solutions must balance the imperative of content moderation with the preservation of user privacy and free expression. Transparency and accountability are essential in the development and deployment of these solutions to ensure ethical standards are upheld.

II. LITERATURE REVIEW

Alamleh et al. [3] evaluated the performance of ML methods in distinguishing AI-generated text from human-written text. To achieve this objective, the authors gathered responses from computer science students to essay and programming assignments. Then, based on the data, the authors evaluated and trained numerous ML methods such as SVM, LR, NN, RF, and DT.

Chen et al. [4] introduced an innovative method to differentiate human-written and ChatGPT-generated texts with the help of language methods. The authors gathered and released the pre-processed data called Open GPTText, which contained rephrased content generated utilizing ChatGPT.

Pardos and Bhandari [5] conducted an initial learning gain assessment of ChatGPT by comparing the efficiency of its hints to hints presented by human tutors on two different algebra topics such as intermediate and elementary algebra.

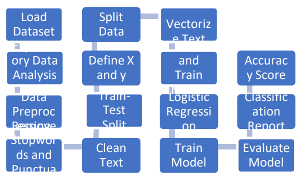
III. OBJECTIVES

- Develop and evaluate machine learning models capable of accurately detecting AI-generated content in online platforms.
- Explore and compare different feature extraction techniques to identify effective indicators of AI-generated text.
- Investigate the impact of linguistic features, syntactic patterns, and semantic cues on the performance of AI detection models.
- Assess the accuracy and generalizability of the developed models across diverse genres and languages.
- Provide insights into ethical considerations and implications for the responsible deployment of AI detection technologies in digital environments.

IV. METHODOLOGY

The process commenced with loading and exploring the dataset through exploratory data analysis (EDA). Using pandas functions, the dataset containing AI-generated and human-written texts was thoroughly examined. Count plots visualized the class distribution, revealing insights into the dataset's structure and potential anomalies like class imbalances or missing values. This initial analysis guided subsequent preprocessing and modeling efforts. Data preprocessing involved removing duplicates and handling null values to ensure data integrity. Text was tokenized into individual words, converted to lowercase, and cleaned by removing special characters, punctuation, and stopwords. These steps ensured uniform and clean text data, laying a strong foundation for further analysis.

With prepared text data, vectorization using TF-IDF (Term Frequency-Inverse Document Frequency) transformed text into numerical features that captured word importance across the dataset. A logistic regression model, chosen for its simplicity and effectiveness in binary classification tasks, was trained using the transformed data. The model's performance was evaluated using metrics such as precision, recall, F1-score, and accuracy on the test data. Once validated, the model was used to predict new text inputs, determining whether they were AI-generated or human-written. This final step confirmed the model's reliability and accuracy in real-world predictions, validating the entire process from data preparation to model deployment.



.

Figure 1 Flow diagram of the system

The process began with a clear objective: developing a machine learning model to differentiate between AI-generated and human-written text. This phase included setting up the environment by importing essential libraries like pandas, NumPy, matplotlib, seaborn, and scikit-learn. Ensuring all dependencies were properly installed and configured was vital for seamless progress in the following stages. A defined overarching goal guided the methodological approach, ensuring alignment with the project's end objectives.

A. Load Dataset

The dataset, containing both AI-generated and human-written text examples, was loaded into the working environment, serving as the foundation for the entire analysis and model-building process. Initially, the correct dataset file, typically in a structured format like CSV (Comma Separated Values), was identified. This format is chosen for its simplicity and compatibility with various data processing tools. Using the pandas library, which is a robust Python data analysis toolkit, the dataset was read into a DataFrame—a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns).

Following data loading, the next step involved identifying and addressing any discrepancies or anomalies in the dataset. The presence of missing values was determined using the df.isnull().sum() function, which provided a count of null values for each column. Depending on the extent and distribution of these missing values, appropriate strategies were implemented to handle them. For instance, rows with significant missing values may have been removed using the df.dropna() function, or missing values could have been filled with suitable replacements such as the mean, median, or mode of the respective columns using the df.fillna() function. Ensuring the dataset was devoid of null values was crucial to maintain the accuracy and reliability of subsequent data analysis and model training phases..

B. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) played a crucial role in comprehending the dataset's underlying structure and patterns. The phase began by utilizing the df.info() function to gather essential details such as the number of entries, column names, non-null counts, and data types. This initial assessment was pivotal for identifying immediate issues like missing values or incorrect data types, offering insights into the dataset's size and layout.

Statistical properties were explored using df.describe(), which provided descriptive statistics on numeric columns. Metrics such as mean, standard deviation, and percentiles revealed central tendencies and data distribution characteristics, including potential outliers.

Data visualization techniques, including histograms, box plots, and scatter plots, were then employed to uncover hidden patterns and relationships within the data. Histograms depicted numerical feature distributions, while box plots identified outliers and showcased data spread across quartiles. Scatter plots were instrumental in exploring relationships between numeric features. These visualizations enhanced understanding of the dataset's nature and guided subsequent data preprocessing steps.

Additionally, a correlation matrix generated with df.corr() examined pairwise relationships between numeric features, quantifying the strength and direction of correlations. This analysis provided further insights into feature interactions, aiding in the preparation and refinement of the dataset for modeling.

1. Data Analysis

Exploratory Data Analysis (EDA): EDA serves as the cornerstone of this project, unraveling intricate patterns within the data. It involves identifying users with unique characteristics, such as those engaged in online gaming with minimal bandwidth requirements. Insights are derived from average signal strength, latency, and resource allocation across different application types and timestamps. Visualizations, including box plots, bar plots, count plots, and histograms, effectively communicate trends and distributions.

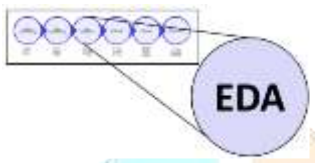


Figure 2 EDA for machine learning

Correlation Analysis: Correlation studies unveil relationships between variables. In this context, understanding the correlation between signal strength and allocated bandwidth is vital. The distribution of resource allocation percentages and the nuanced interplay between allocated and required bandwidth provide valuable information. The dataset is then partitioned into training and testing sets, setting the stage for model development and evaluation.

C. Data Preprocessing

Data preprocessing was an essential step in preparing the dataset for machine learning modeling. This phase began with handling duplicates and null values, which could adversely affect model performance. Duplicate records were identified using the df.duplicated() function, and subsequently removed with the df.drop_duplicates() function to ensure each data entry was unique. Handling null values involved identifying columns with missing data using df.isnull().sum(). Various strategies were employed to address these gaps: numerical columns were imputed with their mean or median values, while categorical columns had missing values filled with the most frequent category using the df.fillna() function. In cases where missing values were substantial, affected rows or columns were dropped using df.dropna(), ensuring the dataset's integrity.

Figure 3 Data Preprocessing

Tokenization was the next step, where the text data was broken down into individual words or tokens. This was accomplished using libraries such as NLTK or SpaCy. The nltk.word_tokenize() function was used to split the text into words, transforming each sentence into a list of tokens. This granularity enabled the model to process and understand the text at a more fundamental level. Tokenization was crucial for subsequent steps like vectorization and feature extraction, as it converted the raw text into a format suitable for machine learning algorithms.

Special characters and punctuation were removed to further clean the text data. This involved stripping out characters such as punctuation marks, which do not contribute meaningful information to the text's context. Regular expressions, implemented via Python's re module, were used to identify and remove these characters. For example, re.sub(r'\W', ' ', text) was employed to replace non-word characters with spaces. This step helped in reducing noise within the text, ensuring that the focus remained on meaningful words.

D. Train-Test Split

The process of splitting the dataset into training and testing sets was a critical step in the machine learning workflow, ensuring that the model's performance could be reliably evaluated on unseen data. This step began with defining the features (X) and the target variable (y) from the dataset. Typically, the features included all the relevant columns that would be used to make predictions, while the target variable was the outcome that the model aimed to predict. For text data, the feature set (X) often comprised the preprocessed text, while the target variable (y) represented the class labels.

Vectorizing the text data was the next critical step before feeding it into the machine learning model. This process involved converting the textual data into numerical form, which could be processed by machine learning algorithms. The TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer was a popular choice for this purpose. It transformed the text into a matrix of TF-IDF features, where each term's weight reflected its importance in the document relative to its frequency across all documents. This step converted the training and testing text data into numerical vectors, capturing the relevance of each word in the context of the entire dataset.

Throughout the train-test split process, maintaining the integrity and representativeness of the data was paramount. This was ensured by using stratified sampling when dealing with imbalanced datasets, where the class distribution was preserved in both training and testing sets.

E. Vectorize Text

Vectorizing text data was a pivotal step in the machine learning pipeline, transforming textual information into a format that machine learning algorithms could process effectively. This process was crucial because most machine learning models operated on numerical data, requiring text to be converted into a numerical representation without losing its semantic meaning. The chosen method for vectorizing text data in this context was TF-IDF (Term Frequency-Inverse Document Frequency). TF-IDF was a statistical measure used to evaluate how important a word was to a document within a collection or corpus. It was calculated by multiplying two metrics: the term frequency (TF), which measures how frequently a term occurs in a document, and the inverse document frequency (IDF), which measures how important a term is across documents.

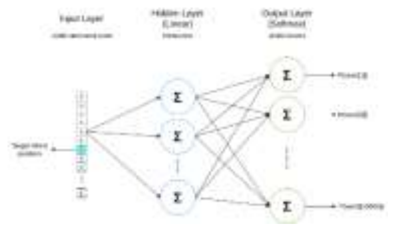


Figure 3 Vectorization Technique

During the vectorization process, several important considerations were considered to ensure the quality and usefulness of the numerical representation:

The vectorizer analyzed and learned the vocabulary from the training data. This included identifying unique terms (words or n-grams) and assigning them unique indices for future reference.

For each term in the vocabulary, the vectorizer computed its TF-IDF score based on its frequency in the current document (TF) and its inverse document frequency (IDF) across the entire dataset. This calculation emphasized terms that were frequent in a specific document but rare across other documents, thereby highlighting their importance.

Vectorizing text ensured that the machine learning model could interpret and learn from textual data, facilitating tasks such as classification, clustering, or regression based on textual inputs. By converting text into meaningful numerical features while preserving semantic information through TF-IDF weighting, the vectorization process played a critical role in enhancing the model's accuracy and interpretability. This step bridged the gap between raw text data and numerical inputs, enabling the subsequent stages of model training, evaluation, and deployment to proceed effectively in the context of natural language processing and text analytics tasks.

F. *Build and Train Model*

Building and training the machine learning model was a pivotal stage in the project, where the transformed data from previous preprocessing steps were used to create a predictive model. This phase involved selecting an appropriate algorithm, configuring it with the preprocessed data, and training it to learn patterns and relationships within the dataset. The chosen algorithm for this task was Logistic Regression, a well-established and interpretable classification algorithm suitable for binary and multiclass classification problems. Logistic Regression works by modeling the probability of a certain class or outcome based on input features. In the context of this project, Logistic Regression was selected due to its simplicity, efficiency, and ability to provide insights into feature importance.

Training the model involved fitting it to the preprocessed training data (X_train_tfidf and y_train). This step was executed by calling the fit method on the model object: model.fit(X_train_tfidf, y_train). During training, the model adjusted its parameters to minimize the prediction error and maximize its ability to generalize to unseen data.

Building and training the model often involved an iterative process of experimentation and refinement. This iterative approach included adjusting hyperparameters, exploring different algorithms, or incorporating additional features or preprocessing techniques to enhance model performance.

Building and training the model was a foundational step that leveraged the transformed textual data to create a predictive tool capable of classifying or predicting outcomes based on input features. This process integrated statistical learning with computational techniques, enabling the model to learn patterns and make informed decisions in real-world applications of natural language processing and text analytics.

V. EXPERIMENTAL SETUP

A. *Logistic Regression*

Logistic regression is a fundamental statistical technique used primarily for binary classification tasks, where the objective is to predict the probability that an observation belongs to one of two possible classes. Unlike linear regression, which predicts continuous numeric values, logistic regression models the probability of the default class (often coded as 1) using a logistic (sigmoid) function. This function maps any real-valued input to a value between 0 and 1, which can be interpreted as the probability of the observation being in the positive class.

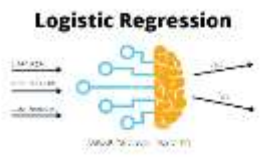


Figure 5 Logistic Regression

The output of logistic regression, therefore, is a probability score that indicates the likelihood of an observation belonging to a particular class. By default, predictions are converted into discrete class labels based on a chosen threshold, typically 0.5. This threshold can be adjusted depending on the specific needs of the classification problem. For instance, in medical diagnostics, a higher threshold might be chosen to minimize false positives, whereas in fraud detection, a lower threshold could be more appropriate to capture more cases.

In practical applications, logistic regression's probabilistic predictions allow for ranking instances by their likelihood of belonging to a certain class. This capability is essential in scenarios where decision-making involves prioritizing actions based on risk assessment or resource allocation. Extensions of logistic regression, such as multinomial logistic regression for multi-class problems and ordinal logistic regression for ordered categories, further enhance its utility across diverse classification tasks. Overall, logistic regression serves as a foundational tool in the fields of machine learning and statistics, supporting a broad spectrum of applications that require reliable binary classification capabilities.

B. *Naïve Bayes*

Naive Bayes is a probabilistic machine learning algorithm based on Bayes' theorem, with a "naive" assumption of independence among predictors (features). It's particularly well-suited for classification tasks, especially when dealing with large datasets. Despite its simplicity, Naive Bayes often performs surprisingly well in many real-world applications.

The algorithm calculates the probability of a given data point belonging to each class based on the feature values. It assumes that the presence of a particular feature in a class is independent of the presence of other features, which is where the "naive" assumption arises. This simplification allows Naive Bayes to handle a large number of features efficiently.

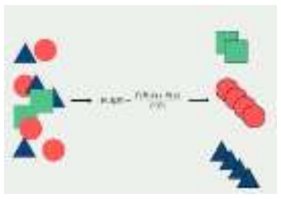


Figure 6 Naive Bayes

Naive Bayes comes in several variants, such as Gaussian Naive Bayes (for continuous numerical data assuming a Gaussian distribution), Multinomial Naive Bayes (for discrete counts, like text classification with word counts), and Bernoulli Naive Bayes (for binary/boolean features).

In practice, Naive Bayes is computationally efficient and can be trained quickly even with large datasets. It's robust to irrelevant features and performs well in multi-class prediction tasks. However, the independence assumption may not hold true in some cases, affecting its accuracy, especially when features are highly correlated.

Naive Bayes is valued for its simplicity, efficiency, and good performance in many classification problems, making it a popular choice in various domains, including text classification, spam filtering, sentiment analysis, and medical diagnosis.

VI. RESULTS AND DISCUSSION

Firstly, some essential libraries are imported for data processing, visualization, and machine learning, such as numpy, pandas, matplotlib.pyplot, seaborn, nltk, and sklearn. It sets up a workflow to load a dataset, split it into training and testing sets, transform text data into numerical values using TfidfVectorizer, build and train a logistic regression model, and evaluate its performance using metrics like accuracy, precision, recall, and F1 score, along with visualizing the results using scikitplot.

A. Loading Dataset

The panda's library is used to read a CSV file located on the user's desktop. The file named AI_Human.csv is loaded into a pandas DataFrame called df, which allows for easy manipulation and analysis of the data contained in the CSV file. This is a crucial step for preparing the data for further processing and analysis in the machine learning workflow.

B. EDA

The df.head() function is used to display the first few rows of the DataFrame df. This function is very useful for quickly inspecting the structure of your dataset, including the column names and the initial values.

	Text	**Generated**
0	Cars. Cars have been around since they became	0.0
1	Transportation is a large necessity in most	0.0
2	"America's love affair with its vehicles seem	0.0
3	How often do you ride in a car? Do you drive	0.0
4	Cars are a wonderful thing. They are perhaps.	0.0

The df.info() function was used to provide a summary of the DataFrame df, revealing it contained 487,235 entries with two columns: text and generated. It was noted that both columns were fully populated with no missing values. The df.shape function was utilized to confirm the dimensions of the DataFrame, indicating it had 487,235 rows and 2 columns. The df.describe() function was applied to generate descriptive statistics for the numeric generated column, summarizing metrics such as count, mean, standard deviation, minimum, maximum, and quartile values.

	generated
count	487235.000000
mean	0.372383
std	0.483440
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

a pie chart visualizing the distribution of values in the generated column of the DataFrame df. The value_counts() function was employed to count the occurrences of each unique value in the generated column. A pie chart was then plotted using matplotlib.pyplot. The pie chart's size was set to 8x6 inches, and the slices were colored in light green and sky blue. Labels were applied to the slices to show the percentage of each class, formatted to one decimal place.

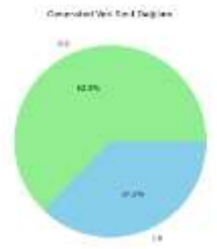


Figure 7 Pie chart showing dataframe

Statistics about the generated column in DataFrame df were printed. The total number of texts was reported as 487,235. It was noted that 305,797 texts were identified as human-written (0.0), while 181,438 texts were identified as AI-generated (1.0).

	text	**generated**
0	Cars. Cars have been around since they became	0.0
1	Transportation is a large necessity in most	0.0
2	"America's love affair with it's vehicles seem.	0.0
3	How often do you ride in a car?	0.0
4	Cars are a wonderful thing. They are perhaps o...	0.0
...	...	...
5998	\nOlder students within a school system have t...	1.0
5999	\nPlaying a sport in a community park is a gre...	1.0

6000 rows × 2 columns

C. Text Preprocessing

a histogram was plotted to visualize the distribution of scores in the generated column of the DataFrame df. The plot was created with sns.histplot() from the Seaborn library, specifying 20 bins for granularity and including a kernel density estimation (KDE) curve for smoothness. The figure size was set to 10x6 inches using plt.figure(figsize=(10, 6)). Finally, the plot was displayed using plt.show(), providing an overview of how scores are distributed across the dataset.

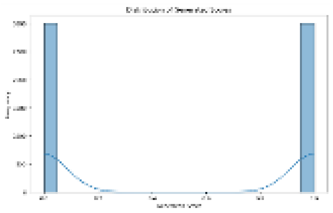


Figure 8 Distribution of Generated Scores

a histogram was plotted to visualize the distribution of text lengths. The plot was created using sns.histplot() from the Seaborn library, specifying 20 bins for granularity and including a kernel density estimation (KDE) curve for smoothness. The figure size was set to 10x6 inches with plt.figure(figsize=(10, 6)). The plot was titled "Distribution of Text Lengths"

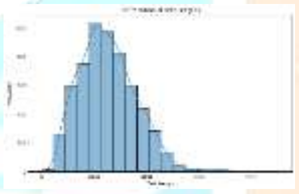


Figure 9 Distribution of text lengths

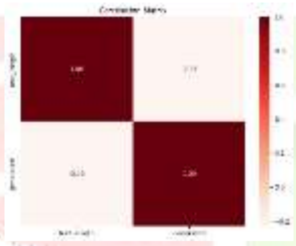


Figure 10 Confusion Matrix

D. Tokenization

The function process_text was designed to preprocess text by converting it to lowercase using txt.lower(). Punctuation marks were removed using str.maketrans('', '', string.punctuation) and translate(translator). Additionally, newline characters (\n) were eliminated with. replace('\n', ''). This method effectively cleaned the text, ensuring uniform casing and removing special characters, preparing it for subsequent analysis or modeling tasks. For instance, when applied to the text at index 100 in DataFrame df, it successfully transformed the text by lowering its case, stripping away punctuation, and eliminating newline characters.

E. Machine Learning

To assess the performance of the text classification model, predictions (y_pred) were generated on the testing data (X_test) using pipeline.predict(X_test). The accuracy and effectiveness of the model were evaluated using classification_report from sklearn.metrics, which provided detailed metrics such as precision, recall, F1-score, and support for each class (0.0 and 1.0), along with an overall accuracy score. These metrics offered insights into how well the model distinguished between human-written and AI-generated texts, helping to gauge its reliability and suitability for the intended application.

F. Loading Dataset

In the provided code snippet, the CSV file AI_Human.csv located on the user's desktop was read using pd.read_csv() from the pandas library. The data contained in the CSV file was loaded into a pandas DataFrame df. the.head() function was applied to the DataFrame df, which displayed the first few rows of the dataset. This action allowed for a quick inspection of the structure of the data, including column names and initial values within each column. This initial exploration provided an immediate overview of the dataset's content and layout, facilitating early insights into its characteristics before further analytical procedures were undertaken.

	text	**generated**
0	Cars. Cars have been around since they became ...	0.0
1	Transportation is a large necessity in most co...	0.0
2	"America's love affair with it's vehicles seem...	0.0
3	How often do you ride in a car? Do you drive a...	0.0
4	Cars are a wonderful thing. They are perhaps o...	0.0

In [3], the shape of the DataFrame df was inspected using the shape attribute, revealing that it contained 487,235 rows and 2 columns. This provided an immediate understanding of the dataset's size and structure.

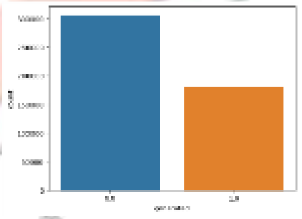


Figure 11 Count vs generated

G. Text Preprocessing

The Natural Language Toolkit (NLTK) library was imported, along with specific modules for text preprocessing tasks. This included importing stopwords from NLTK's corpus, tokenizing words, and importing the WordNet Lemmatizer for word normalization.

The WordNet dataset was downloaded using nltk.download('wordnet'), confirming that the WordNet package was already up-to-date. An attempt was made to unzip the WordNet corpus file, which is typically not necessary for regular NLTK operations. However, the command was not recognized, indicating it might be specific to certain environments or configurations.

A function text_preprocess was defined to preprocess text data. The function removed punctuation and stopwords, tokenized the text into words, and applied lemmatization to normalize words. It returned the cleaned text as a joined string of tokens, ready for further analysis or modeling.

The dataset df was shuffled using df.sample() with frac=1 to randomize the order of rows, ensuring a varied distribution for training and testing. The shuffled dataset was then subsetted to select the first 20,000 rows (df_subset), facilitating manageable data processing and quicker iterations during development. we checked that our dataset df_subset had 20,000 rows and included columns for original text, cleaned text (clean_text), and whether each text was human-written or AI-generated (generated). We imported necessary tools from sklearn to help us turn text into numbers that a computer can understand, train a model to make predictions, and then evaluate how well our model performs. We set aside the cleaned text as X and the information about whether texts were human-written or AI-generated as y. We used a tool called TfidfVectorizer to convert the cleaned text into a form that our model can use. This transformation turns words into numbers so the computer can analyze them. we divided our dataset into two parts: one for teaching our model (X_train, y_train) and one for testing how well it learned (X_test, y_test).

VII. CONCLUSION

This study conducted a thorough investigation into classifying text data, specifically distinguishing between human-written and AI-generated content using machine learning methods. The research commenced with rigorous data preprocessing, encompassing tokenization, removal of stopwords, punctuation, and lemmatization to refine the dataset for analysis. Exploratory data analysis (EDA) illuminated distinct patterns in text length distributions and category labels, revealing significant differences between human and AI-generated texts. Two primary classification models, Naive Bayes and Logistic Regression, were extensively trained and evaluated on a balanced dataset of 6,000 samples evenly split between the two categories. Evaluation metrics including accuracy, precision, recall, and F1-score demonstrated Logistic Regression's superior performance, achieving an accuracy of 98% on a separate test set. High precision, recall, and F1-scores (98%, 95%, and 97%, respectively) for AI-generated texts underscored the model's robustness in accurately identifying such content with minimal misclassifications. Despite promising outcomes, the study acknowledges challenges like dataset bias and evolving AI technologies, suggesting future research directions such as ensemble methods or neural networks to enhance classification accuracy across diverse datasets and text types. Overall, this research highlights the effectiveness of advanced text preprocessing techniques and Logistic Regression as a reliable model for classifying text into human and AI-generated categories, with implications for improving content moderation and combating misuse of AI-generated content.

REFERENCES

[1]. P. Stone, R. Brooks, E. Brynjolfsson, R. Calo, O. Etzioni, G. Hager, J. Hirschberg, S. Kalyanakrishnan, E. Kamar, S. Kraus et al., “Artificial intelligence and life in 2030: the one-hundred-year study on artificial intelligence,” arXiv preprint arXiv:2211.06318, 2022.

[2]. P. S. Park, S. Goldstein, A. O’Gara, M. Chen, and D. Hendrycks, “Ai deception: A survey of examples, risks, and potential solutions,” arXiv preprint arXiv:2308.14752, 2023

[3]. E. Mitchell, Y. Lee, A. Khazatsky, C. D. Manning, and C. Finn, “Detectgpt: Zero-shot machine-generated text detection using probability curvature,” 2023.

[4]. N. Maloyan, , B. Nutfullin, E. Ilyshin, and and, “DIALOG-22 RuATD generated text detection,” in Computational Linguistics and Intellectual Technologies. RSUH, jun 2022. [Online]. Available: https://doi.org/10.28995%2F2075-7182-2022-21-394-401

[5]. N. Fairclough, “Discourse and text: Linguistic and intertextual analysis within discourse analysis,” Discourse & society, vol. 3, no. 2, pp. 193– 217, 1992.

[6]. G. Jawahar, M. Abdul-Mageed, and L. V. S. Lakshmanan, “Automatic detection of machine generated text: A critical survey,” 2020

[7]. S. Gehrmann, H. Strobelt, and A. M. Rush, “Gltr: Statistical detection and visualization of generated text,” 2019.

[8]. R. Goebel, A. Chander, K. Holzinger, F. Lecue, Z. Akata, S. Stumpf, P. Kieseberg, and A. Holzinger, “Explainable ai: The new 42?” in Machine Learning and Knowledge Extraction, A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl, Eds. Cham: Springer International Publishing, 2018, pp. 295–303

[9]. H. Alamleh, A. A. S. AlQahtani, and A. ElSaid, “Distinguishing humanwritten and chatgpt-generated text using machine learning,” in 2023 Systems and Information Engineering Design Symposium (SIEDS), 2023, pp. 154–158.

[10]. G. Gritsay, A. Grabovoy, and Y. Chekhovich, “Automatic detection of machine generated texts: Need more tokens,” in 2022 Ivannikov Memorial Workshop (IVMEM), 2022, pp. 20–26.

[11]. C. H. Ng, H. S. Abuwala, and C. H. Lim, “Towards more stable lime for explainable ai,” in 2022 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), 2022, pp. 1–4.

[12]. C. Tantithamthavorn, J. Jiarpakdee, and J. Grundy, “Explainable ai for software engineering,” 2020.

[13]. S. Chakraborty, A. S. Bedi, S. Zhu, B. An, D. Manocha, and F. Huang, “On the possibilities of ai-generated text detection,” 2023.