AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Projekt dyplomowy

Comparative Evaluation of Methods for Classifying Human-Written and AI-Generated Text

Analiza porównawcza metod klasyfikacji tekstów pisanych przez człowieka i generowanych przez sztuczną inteligencję

Autor: Wojciech Neuman
Kierunek studiów: Informatyka i Systemy Inteligentne
Opiekun pracy: dr inż. Krzysztof Kluza

Kraków, 2025

ABSTRACT

With the rapid growth and adaptation of AI-generated text produced by Large Language Models (LLMs) such as ChatGPT, distinguishing between human-written and machine-generated content has become an urgent challenge. This thesis explores and evaluates various machine learning and deep learning text classifiers, focusing on both classification performance and interpretability.

The study begins with foundational models such as Logistic Regression and Support Vector Machines (SVM) based on traditional text representation methods like Bag-of-Words (BoW) and TF-IDF. Progressively more advanced architectures, including Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory networks (BiLSTMs), are explored using pre-trained fastText word embeddings to capture semantic and sequential patterns. Finally, transformer-based model BERT is introduced as a current state-of-the-art solution in Natural Language Processing (NLP).

A comparative analysis evaluates the models and text representations, with the goal of identifying the most effective approaches, while also analyzing their trade-offs in terms of complexity, accuracy, and computational cost. To make our models easier to understand and build trust in their predictions, Explainable AI (XAI) methods like SHAP and LIME are used to show how the models make decisions and which features are most important. The results not only highlight the best-performing methods but also offer guidance on how to make future text classification models more interpretable.

Contents

1 Introduction

Intelligence is one of the greatest virtues since the dawn of time. The ability to learn, understand, and make judgments or have opinions that are based on reason, was the thing that made our species so special and allowed our civilization to succeed. However, ever since computers were created in the mid-20th century, suddenly it became possible to simulate thinking, problem-solving, even natural language: Artificial Intelligence was born [1]. It is the science and engineering of making intelligent machines, especially intelligent computer programs [2].

There has always been great debate over whether computers can reach a full Artificial General Intelligence – a system that can do anything a human can do [3], even in emotional, social or musical spectrum. Those who agree that it is possible certainly have more arguments for it after rapid growth of Large Language Models (LLMs), particularly the most famous created by OpenAI – called ChatGPT. These models are trained on massive amounts of text data and are able to generate human-like text, answer questions, and complete other language-related tasks with high accuracy. Steve Jobs, one of the greatest minds in the history of modern technology said this in 1985 about building a tool that can answer our questions like the best teacher: "My hope is someday, we can capture the underlying worldview of Aristotle – in a computer. And someday, some student will be able not only to read the words Aristotle wrote, but ask him a question – and get an answer.". Today, after less than 40 years, this dream came into reality and we are all blessed to experience it, having tools such as ChatGPT acting as a modern embodiment of Aristotle's wisdom.

These tools not only enhance learning opportunities but also revolutionize teaching methodologies. They can automate increasingly complex tasks, scan the internet at lightning

speed and pinpoint precision. Its utility can be shared across all students in the world, from kindergarten to esteemed professors, and across all professions – from IT specialists and medical diagnosticians to paralegals and beyond.

However, with the great power comes a great responsibility – this rapid adaptation of AI-generated text comes with a plenty of issues. LLMs can never be fully trusted upon, as they can be often biased – their functioning is based mostly on statistics, which may lead to a lack of true understanding and expertise. There is also an issue with distinguishing between real human text and generated human-like text [4]. These issues raises concerns across various domains, one most prominent is education. Beyond academic concerns, AI-generated text has been misused in various other contexts such as spreading spam across the internet. This includes junk emails or social media bots distributing malicious content or links, raising further challenges in ensuring online security and authenticity.

Objectives and Workflow

The challenges that arise after this massive adoption of AI-generated text, as described above, call for effective solutions in order to distinguish between human-written and machine-generated content. One promising approach is to build classifiers capable of addressing this problem using various machine learning techniques. This thesis focuses on developing, analyzing, and comparing these classifiers.

In the following chapter (Chapter 2), we present the technical background of the project, including dataset preparation and a detailed explanation of the tools and technologies used. After this, in Chapter 3, we create five distinct models of varying complexity. We begin with simpler approaches, based in classic machine learning and natural language processing (NLP) techniques, such as Bag of Words and Logistic Regression, followed by TF-IDF (Term Frequency-Inverse Document Frequency) paired with Support Vector Machines. These foundational methods, while being effective in most cases, often struggle to capture the nuances and context within texts.

As the complexity increases, the workflow transitions to more advanced models, including a Convolutional Neural Network (CNN) model and a Bidirectional Long Short-Term

Memory (BiLSTM) model, paired with word embeddings. These models are better suited for understanding sequential patterns in texts. Finally, transformers are introduced as the state-of-the-art approach for most of the NLP tasks, including text classification, due to their high ability to capture context within paragraphs.

After carefully designing these models, in Chapter 4 we conduct a comparative analysis of their performance and results. We want to show the strengths, if any, of each model, but also focus on their limitations and examples where they achieve low performance. From this, the goal is to identify the most reliable model that achieves the highest accuracy and can be relied upon for practical use.

Given the inherent complexity of some of these models that at some point is no longer comprehensible to humans, explainability becomes crucial. We want to find out what is inside of the "Black Box", for that, in Chapter 5 we are going to use Explainable Artificial Intelligence (XAI). This technique aims to make AI behavior more intelligible to humans by providing explanations [5]. Specifically, SHapley Additive exPlanations (SHAP) and Local Interpretable Model-agnostic Explanations (LIME) will be employed to analyze and interpret model predictions.

The final chapter (Chapter 6) will conclude the thesis with a summary of the study and future directions. Following this workflow, this paper not only addresses the critical challenge of distinguishing between human and AI-generated text but also builds the foundation for future advancements in text classification by creating and comparing multiple models. Additionally, by emphasizing the importance of explainability, we ensure that the models are not only effective but also interpretable.

2 Technical Background

This chapter provides an overview of the features needed for the classification task, focusing on the selection and analysis of the most suitable dataset. Initial data analysis highlights a major difference between human-written and AI-generated content, emphasizing the importance of considering this when evaluating classifiers. The chapter then reviews the tools and technologies that support the development of the models, including programming languages, machine learning libraries, and computational resources.

Entire code base used in this research, including data processing scripts, model training code, and evaluation metrics, is publicly available on GitHub [6]. This repository contains all necessary resources for reproducing and extending this study.

2.1 Data understanding

To properly train effective classifiers, a well-balanced dataset with high-quality data is essential. Ideally, a training set is a representative subset of the population. In practice, we rarely achieve a perfectly representative set, as we often rely on random samples [7]. This challenge is even more prominent in the context of this paper, where the number of possible sentences is virtually infinite, let alone entire paragraphs or even longer pieces of text.

After carefully reviewing possible dataset options, OpenLLMText [8] stood out, due to its substantial size, diverse data sources, and the structured correlation between human-written and AI-generated texts. In this dataset, most prompts for generating AI text were human-written paragraphs, with the AI tasked to rephrase them. This set of data contains more than 350,000 text entries collected from 5 different sources:

- Human-written texts: around 70,000 entries randomly selected from the OpenWebText dataset, particularly from user-generated content from Reddit before 2019.
- ChatGPT-generated texts: around 70,000 entries created by GPT-3.5 Turbo, each being a paragraph-by-paragraph rephrasing of the human-written data.
- LLaMA-7B-generated texts: around 70,000 entries generated by the LLM Meta AI (LLaMA-7B) using similar paragraph-by-paragraph rephrasing techniques.
- PaLM-generated texts: around 70,000 entries produced by the Pathway Language Model (text-bison-001), also rephrasing human-written data.
- GPT2-output dataset: around 80,000 entries adapted from the GPT2-output dataset, released by OpenAI (GPT2-XL).

Google, the creator of PaLM, has announced that starting April 9, 2025, this model will not be accessible as it will be migrated to Gemini models. As a result, PaLM-generated data is excluded from our model training. Similarly, we omit using GPT-2 due to its lack of alignment with human-generated paragraphs and its outdated release (2019).

While GPT-3.5 Turbo and LLaMA-7B are not the most recent models – the first one was released on March 14, 2023, and the second on July 18, 2023 – they still offer high utility. Their alignment with human-written texts through paragraph-by-paragraph rephrasing is something unique that could not be found with newer models such as GPT-4 or Gemini.

Ensuring a balanced dataset for creating and evaluating our models involves using entries from the human-written subset while alternately selecting corresponding entries from GPT-3.5 Turbo and LLaMa. By doing so, we guarantee an equal class ratio of 1, resulting in a total of 134,886 entries. The dataset is then divided into three subsets [7]:

- Training set (approximately 75% of all entries) – a set used for learning and estimating parameters of the model.
- Validation set (approximately 15% of all entries) – a set used to evaluate the model, usually for model selection.

- Testing set (approximately 10% of all entries) – a set of examples used to assess the predictive performance of the model.

Managing such a large dataset can be challenging during model training. To address this, we create sample datasets containing 50,000, 20,000, 5,000, 3,000 and 1,000 entries, each maintaining the 75-15-10 proportion for training, validation, and testing sets, respectively. These sample sets are used in the early stages of model development, allowing us to conserve time and computational resources. This approach enables us to focus on critical aspects such as feature engineering, hyperparameter selection, model architecture tuning, and evaluation metrics, which are essential for optimizing model performance.

Initial data exploration

During initial exploration of the data, we notice a significant difference in the average text length between human-written entries and those generated by AI. Texts created by humans had, on average, 572 words, while their rephrased by AI versions only had 304. During evaluation of the results, this difference must be carefully considered, because our classifiers should not rely on text length. Instead, they should focus on other features, such as language style, word choice, and syntactic structures, that differentiate human and AI-generated content. Histograms of these distributions are displayed in Figure 2.1.

The distribution for human text closely follows a positively skewed, unimodal pattern, peaking around 230 words. On the other hand, the AI-generated text is more unpredictable, resembling two overlapping binomial distributions. These two distributions for ChatGPT and LLaMA are shown in Figure 2.2.

Upon closer inspection, it is clear that the ChatGPT distribution follows a typical binomial pattern. However, the LLaMA-generated texts are responsible for the strange two-peak shape in the overall AI text distribution. This uncommon pattern is likely an outcome of LLaMA typically producing shorter outputs for most of the texts, creating the first peak on the left. But when the model is given long inputs, which are fairly common in the dataset, it tries to cut them during rephrasing due to model limitations.

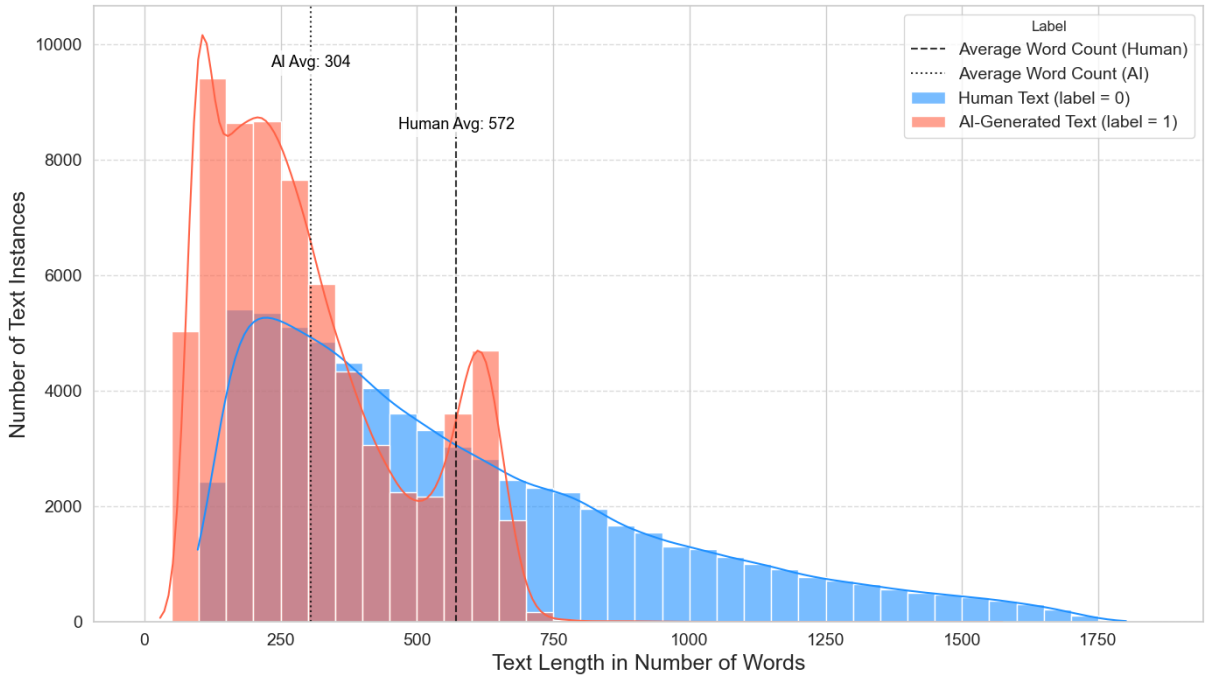


Figure 2.1: Distribution of text length for human and AI-generated text. The human-written text tends to be more consistent, while AI-generated text shows more variation.

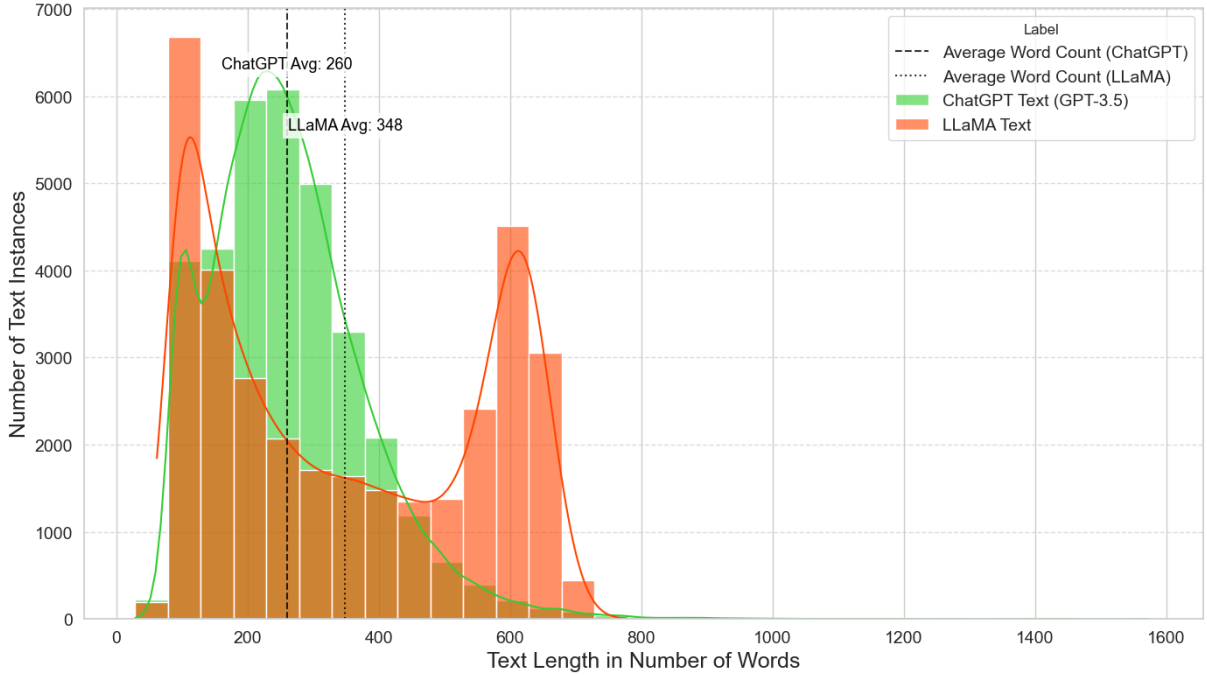


Figure 2.2: Distribution of text length for AI-generated text, showing two peaks for ChatGPT and LLaMA.

2.2 Tools and Technologies

Python is selected as the sole programming language in this research due to its high utility in scripting and extensive access to libraries for data processing and machine learning.

To address the challenges of managing long and complex scripts, Jupyter Notebooks are used. They offer features such as cell-based execution and a clear, interactive interface.

Regarding the libraries utilized, several key ones are worth highlighting:

- **NumPy** – a fundamental library for scientific computing with Python, providing support for arrays, matrices, and a wide range of mathematical functions for data manipulation and analysis.
- **Pandas** – a library that allows easy manipulation of data through its primary structures, Series and DataFrames, which are equivalent to columns and tables, respectively. It enables efficient data loading, processing, and preparation for training or analysis of experimental results.
- **Scikit-learn** – a widely used machine learning library in Python that provides a range of tools for model building, evaluation, and tuning. It is particularly useful during creation of Logistic Regression and Support Vector Machines.
- **PyTorch** – a high-performance deep learning library that offers an imperative and Pythonic programming style, making it easy to debug and maintain. PyTorch supports hardware accelerators such as GPUs, making it extremely well-suited for developing such models as Convolutional Neural Networks (CNNs), Bidirectional Long Short-Term Memory (BiLSTM) and Transformers, which are discussed in this paper.
- **Hugging Face** – a platform and library focused on simplifying the implementation, training, and fine-tuning of transformer-based models such as BERT. The library offers pre-trained weights, tokenizers, and pipelines, which accelerate model deployment and fine-tuning. Hugging Face also provides a collaborative platform where users can share models, datasets, and other machine learning resources, creating a strong community-driven ecosystem.
- **TensorFlow** and **Keras** – TensorFlow is an open-source framework for building and deploying machine learning models, with a focus on deep learning. It offers flexible and powerful tools for model training, evaluation, and deployment. Keras,

now integrated with TensorFlow, provides a high-level API that additionally simplifies the process of designing, training, and experimenting with neural networks. Together, TensorFlow and Keras make it easier to work with complex models like BERT, especially when fine-tuning or deploying them in production environments.

- **Matplotlib** and **Seaborn** – libraries used for data visualization. Matplotlib is a flexible tool for creating a variety of plots, while Seaborn builds on it, providing a higher-level interface for statistical graphics. Together, they enable clear and informative visualizations of model performance and data insights.

- **SHAP** and **LIME** – both SHAP and LIME are widely used XAI libraries for model interpretability. SHAP provides consistent and mathematically grounded feature importance values, explaining the impact of each feature on predictions. LIME, on the other hand, approximates complex models locally with simpler, interpretable models, offering explanations for individual predictions. Both are valuable tools for making black-box models more transparent and understandable.

Since training models on a large dataset containing over 130,000 texts and aiming at high metric results requires significant computational power, Google Colaboratory and its paid Pro subscription is used. The subscription provides access to Tesla T4 and NVIDIA A100 GPUs – two engines extensively used in this study. A total of 300 compute units were purchased during the research. Each Pro subscription costs $12.29 per month and provides 100 compute units.

The Tesla T4 GPU offers 12.7 GB of system RAM and 15 GB of GPU VRAM. It consumes approximately 1.44 units per hour. With its relatively high computational power and low unit consumption, this engine is used during most of the training process and hyperparameter optimization.

The NVIDIA A100 GPU, on the other hand, provides 83.5 GB of system RAM and 40 GB of GPU VRAM. It consumes around 8.47 units per hour, enabling less than 12 hours of computation per subscription. While the A100 GPU is exceptionally powerful, with an estimated cost ranging from $8,000 to $10,000, it is reserved for final computations due to its higher unit consumption.

3 Overview of classification models

This chapter analyzes five text classification models, arranged from the least complex, oldest methods to the newest, state-of-the-art approaches. The progression reflects the evolution of techniques, from simpler frequency-based methods to advanced models that capture deep contextual and linguistic patterns.

Having input text in all of the models discussed, we first preprocess the data by removing unnecessary elements such as punctuation. Additionally, all words are converted to lowercase to ensure consistency and reduce redundancy caused by case differences. Normally, stopwords – common words such as "are", "it", or "you" – are removed in Natural Language Processing (NLP) tasks like text classification or sentiment analysis to focus on more meaningful terms. However, in this case, we find that retaining stopwords produces better results, as they contribute to the model's understanding of the text.

After preprocessing, we perform tokenization for all of the models. It is a process of breaking down a text into smaller units called tokens. In the context of NLP, tokenization refers to splitting a text document into individual words, sentences, or even smaller units like characters or subwords. In our case, we will be splitting this text into individual words.

3.1 Logistic Regression

The first model uses the most straightforward and interpretable approach for classifying text. Following text preprocessing and tokenization, classification pipeline is divided into two main steps:

Vectorization (BoW)

In this step, text data is transformed into a numerical format suitable for machine learning models. We use the Bag of Words (BoW) representation implemented via the `CountVectorizer()` class from the Scikit-learn library. In the BoW model:

- Each document (or sentence) is represented as a sparse vector.
- Each dimension of the vector corresponds to a unique word in the corpus.
- The value in each dimension represents the frequency or presence of that word in the document.

To perform this, we use `CountVectorizer.fit_transform()` method which learns the vocabulary from the training data (by mapping words to indices) and creates mathematical matrix that describes the frequency of terms that occur in each document in a collection (DTM – document-term matrix) by transforming it into the BoW representation.

It is worth highlighting that the product of this vectorization is huge – shape of `X_train_dtm` reached (101284, 547607) – 101284 being the number of samples and 547607 being the number of distinct words found in it.

Logistic Regression Model Creation

For classification we are going to use Logistic Regression (LR). It is a special case of Linear Regression, designed for problems where the output is not continuous but categorical – typically binary, such as 0 or 1. Logistic Regression takes the outcome variable and predicts probabilities by applying a sigmoid function to map these probabilities to binary classes.

In this case, we use the `sklearn.linear_model.LogisticRegression` class from the Scikit-learn library. This class implements logistic regression for binary (or multinomial) classification tasks, offering flexibility with hyperparameters and extensive documentation and resources online.

In order to find the best set of hyperparameters, we are going to use Grid Search, also known as a full factorial design. It is the most basic hyperparameter optimization (HPO) technique. The user specifies a finite set of values for each hyperparameter, and Grid

Search evaluates the Cartesian product of these sets. This results in extremely high computational cost – required number of model trainings grows exponentially with the dimensionality of the configuration space. At the same time, it is not deterministic when it comes to finding the best set of parameters, as it is limited to the values that we provide in the grid.

However, Logistic Regression is a relatively simple model compared to the more complex models that will be discussed later in this project. This simplicity allows us to afford the computational cost of an exhaustive search using Grid Search. To remove the risk of overfitting, we are using cross-validation.

Cross-validation is a technique used to evaluate the performance of a model on unseen data – it divides the available data into multiple folds, using one as a validation set, and the rest for training. Yet again, Scikit-learn provides a solution – the `GridSearchCV` class. The hyperparameter search explores a defined parameter grid to identify the best set of parameters for the `LogisticRegression` model. The Table 3.1 summarizes the hyperparameter ranges considered.

Hyperparameter	**Range Explored**
Penalty	{'l1', 'l2'}
C	{0.1, 1, 10, 100, 1000} (logarithmic scale)
Solver	{'lbfgs', 'liblinear', 'sag'}
Max Iterations	{200, 400, 800, 1600}

Table 3.1: Grid Search Hyperparameter Ranges for Logistic Regression

This implementation results in 450 different fits due to the Cartesian product of all parameter combinations with 5-fold cross-validation. The set with the best hyperparameters found is displayed in Table 3.2.

Hyperparameter	Final Value
C	0.1
Max Iterations	200
Penalty	l1
Random State	42
Solver	liblinear

Table 3.2: Best Hyperparameters for Logistic Regression

3.2 Support Vector Machines (SVM)

The second model introduces a more complex approach for text classification, leveraging advanced techniques to capture the significance of words more effectively. This process comprises the following steps:

Vectorization (TF-IDF)

Similarly to the approach discussed in the previous section, we perform vectorization on the tokenized and preprocessed text. This time, we use a more advanced technique, Term Frequency-Inverse Document Frequency (TF-IDF), to represent the importance of words in the documents. As the term implies, TF-IDF calculates values for each word in a document through an inverse proportion of the frequency of the word in a particular document to the percentage of documents the word appears in. Words with high TF-IDF values indicate that they are both frequent in the document and unique across the corpus, making them more significant for classification [9]. This method consists of three main components:

1. **Term Frequency (TF)**

   The Term Frequency measures how often a term t appears in a document d. It is defined as:

$$\text{TF}(t, d) = \frac{\text{Number of times } t \text{ appears in } d}{\text{Total number of terms in } d}$$

2. **Inverse Document Frequency (IDF)**

   The Inverse Document Frequency measures how unique or significant a term is across the corpus. It is calculated as:

$$\text{IDF}(t) = \log\left(\frac{N}{1 + \text{DF}(t)}\right)$$

   Where:

   - N: Total number of documents in the corpus.
   - $\text{DF}(t)$: Number of documents containing the term t.
   - Adding 1 to $\text{DF}(t)$ prevents division by zero.

3. **TF-IDF Score**

   The TF-IDF score for a term t in a document d is given by:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

To perform this transformation, we use the `TfidfVectorizer` class from the Scikit-learn library and its `fit_transform()` method. This method learns the vocabulary from the training data and computes the TF-IDF values for each term. The result is a sparse matrix where each row represents a document, and each column corresponds to a unique term in the corpus. The matrix contains the TF-IDF values, which are then used as features for classification.

Support Vector Machines Model Creation

SVM is a supervised machine learning algorithm designed to classify data by finding an optimal line or hyperplane that maximizes the margin between different classes in an N-dimensional space. Among the various classification techniques available for data classification, no single method has been found to consistently deliver the best accuracy across all kinds of datasets. SVM is gaining increasing attention because its pure mathematical foundation and proven performance in diverse real-world applications [10].

Once again, for similar reasons highlighted in the previous section, we are utilizing the Scikit-learn library, which provides a comprehensive `sklearn.svm.SVC` class.

To identify the optimal set of hyperparameters, we employ Sequential Model-Based Optimization (SMBO), also known as Bayesian optimization. SMBO is a general-purpose technique for function optimization that is among the most call-efficient methods in terms of function evaluations currently available. This approach leverages probabilistic models to guide the search for the optimal hyperparameters. By focusing on the more promising regions of the search space, SMBO reduces the number of evaluations required, making it particularly suitable for high-dimensional or computationally expensive models where evaluations are costly.

However, despite its crucial advantages, some downsides of Bayesian Optimization must be acknowledged. This method may get stuck in local optima, and finding the absolute optimal set of hyperparameters is not guaranteed, as it remains a heuristic approach.

In this study, we utilize Optuna [11], an open-source hyperparameter optimization framework, to automate hyperparameter search. Optuna allows us to define "studies" with specific objectives – here, we aim to maximize the average cross-validation F1-score. During an initial exploration of various parameter grid with different kernels, along with faster execution, which allows us to explore additional hyperparameters. Based on this observation, we expand the grid only for the linear kernel.

Hyperparameter	**Range Explored**
C	{0.001, 0.01, 0.1, 1, 10, 100} (logarithmic scale)
Tolerance (tol)	{0.00001, 0.001, 0.1} (logarithmic scale)
Max Iterations	{100, 500, 1000, 5000, 10000}
Shrinking Heuristic	{True, False}

Table 3.3: Hyperparameter Ranges Explored for Linear Kernel SVM

During the training process, 5-fold cross-validation is used to evaluate performance of the model on unseen data, dividing our dataset into multiple subsets and rotating validation set. This approach explores a continuous hyperparameter space, unlike traditional grid

search, making it more efficient for larger parameter spaces. The optimization was run for 50 trials, which provided satisfactory results without excessive computational resources.

To standardize our feature space and ensure consistent scaling, a 'StandardScaler' from Scikit-learn library is included in the preprocessing pipeline. The scaling step ensures that all features are normalized to a consistent range, improving the convergence behavior of the linear kernel SVM.

The set with the best hyperparameters found is displayed in Table 3.4:

Hyperparameter	**Final Value**
C	0.433
Tolerance (tol)	1.736×10^{-5}
Max Iterations	6000
Shrinking Heuristic	True
Random State	42

Table 3.4: Best hyperparameters found using Optuna with preprocessing pipeline

3.3 Convolutional Neural Networks (CNNs)

Although Convolutional Neural Networks (CNN) are best known for their success in computer vision field, their architecture design also enables very effective performance in natural language processing (NLP). Their effectiveness arises from their convolutional layers having power of extracting local features from data and accumulating global information by building hierarchical structures [12]. In other words, the CNN model can detect context well.

The architecture of the model is based on the design outlined in 2014 by Kim [13] and further refined in 2016 by Zhang [14]. A visualization of Zhang's architecture is shown in Figure 3.1.

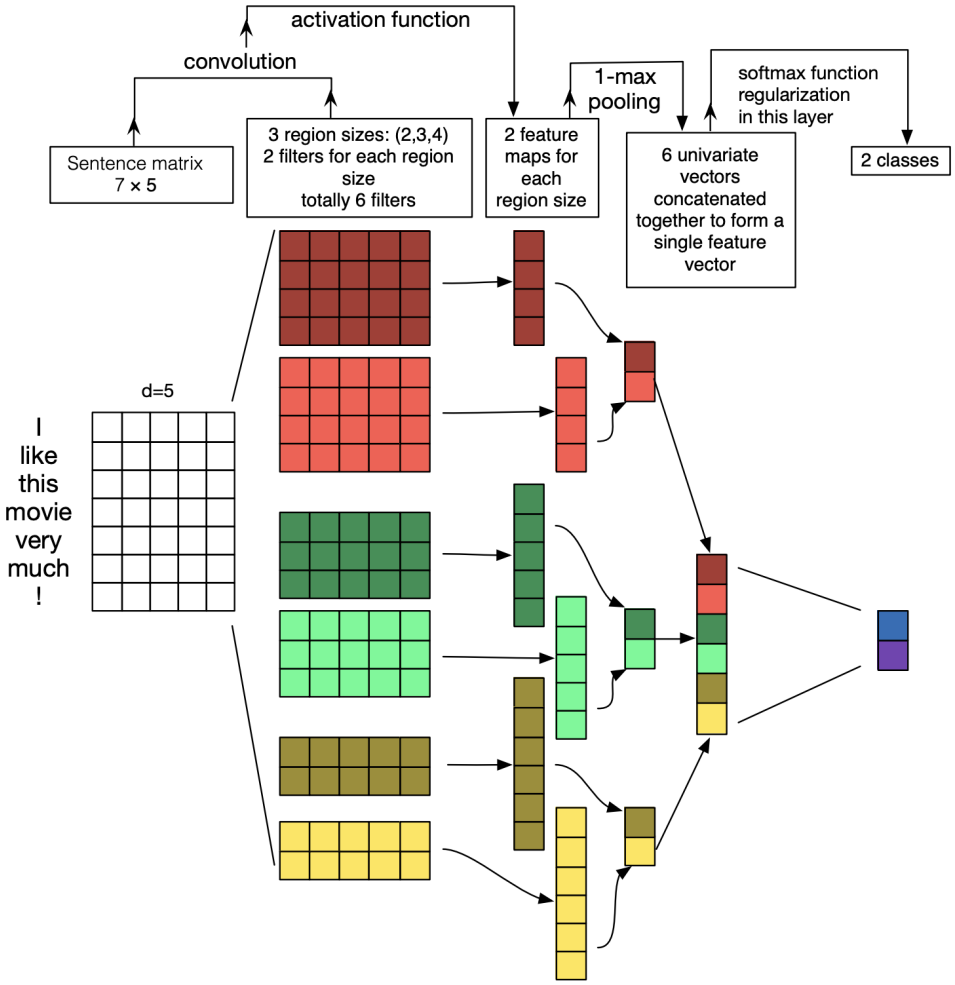


Figure 3.1: CNN architecture for text classification [14].

Preprocessing and Embedding for CNN Input

Even though training of this model is highly complex, we could leverage the NVIDIA A100 GPU with 40 GB of GPU VRAM and 83.5 GB of system RAM to efficiently process computations on the entire dataset.

The data undergoes preprocessing, as described Section 3. Next, tokenization is performed. During this process, we determine the maximum sentence length and pad each tokenized sentence with the '<pad>' token if it is shorter than the max_len. If a token is not found during the testing phase, it is assigned the '<unk>' token. The result of this encoding process is a dictionary mapping each word to its corresponding index.

After obtaining the tokenized output, we use it to create word embeddings. Word embeddings are projections in a continuous space of words designed to preserve the semantic and

syntactic similarities between them [15]. Instead of Word2Vec suggested in the reference paper, our key modification is using FastText embeddings [16]. These embeddings contain 2 million word vectors that were trained with 600 billion tokens and are newer and larger version of Word2Vec. This choice is supported by the work described in [17], a public article that is a basis for this model implementation. We select an embedding dimension of 300, meaning that each word is represented by a 300-dimensional vector.

Using these embeddings, each text input is transformed into a sentence matrix, where each row corresponds to the word embedding of a token in the sentence. This matrix has a $s \times d$ dimensionality, where s is the length of the input (always equal to the longest training document due to padding) and d is the size of word embedding – 300.

The final step involves converting the datasets into PyTorch tensors, which is essential for using the this library. We then create TensorDataset objects to group the inputs and labels together. After that, we prepare the data for training and validation by creating DataLoader instances, which handle batching, shuffling, and efficient data loading. The training data is shuffled to ensure randomness during training, while the validation data is not shuffled. The `DataLoaders` are then returned for use during model training and evaluation.

CNN Model creation

The model implemented is a 1-dimensional Convolutional Neural Network (CNN), as shown in Figure 3.1. The architecture begins with an embedding layer, followed by multiple 1D convolutional layers using filters of varying sizes. These layers are designed to capture local patterns in the input sentences effectively.

In the research, various filter region sizes were examined, ranging from a few filters (1–4 regions) to different sizes (2–16). All configurations demonstrated similar accuracy. However, a combination of filter sizes of 2, 3, 4, and 5 achieved the best performance. For example, consider the sentence:

"Life is like a box of chocolates."

Filters of varying sizes can extract different n-gram features from this sentence, as shown below:

- **Filter size 2**: Extracts bigrams, e.g., *["Life is", "is like"]*.
- **Filter size 3**: Extracts trigrams, e.g., *["Life is like", "is like a"]*.
- **Filter size 4**: Extracts 4-grams, e.g., *["Life is like a", "is like a box"]*.
- **Filter size 5**: Extracts 5-grams, e.g., *["Life is like a box", "is like a box of"]*.

Additionally, the number of feature maps for each filter region size is carefully examined. Researchers observed that this number should ideally range between 100 and 600, as exceeding 600 features significantly increases the risk of overfitting. After experimenting with various values for the number of feature maps, we select 200, 300, 400, and 500 feature maps for filter sizes of 2, 3, 4, and 5, respectively, as the optimal configuration for our model.

The convolutional layers are followed by a max pooling operation, to reduce the dimensionality and retain the most important features. The pooled features are then passed through a fully connected layer, followed by a dropout layer with a dropout rate of 0.6 to mitigate overfitting, an issue commonly encountered during training. For optimization, we use Adadelta, a stochastic optimization technique that allows for per-dimension learning rate method for stochastic gradient descent (SGD). Finally, the model outputs logits for each class in the classification task.

The model is trained with a learning rate of 0.1 over 30 epochs. Additionally, an early stopping mechanism is added. If the validation F1-score does not improve over the last three epochs, we end the process, and the model state with the highest F1-score is saved.

3.4 Bidirectional Long Short-Term Memory (BiLSTM): An Advanced RNN Architecture

Recurrent Neural Networks (RNNs) are a class of neural networks designed specifically for sequential data. In contrast to CNNs, RNNs are unable to extract features in a parallel way. Their bidirectional nature allows them to better capture dependencies between elements in a sequence, making them suitable for tasks such as text classification. On the other hand, they raise significant challenges when dealing with long sequences, such as vanishing or exploding gradient problem, which limits their ability to learn long-term dependencies.

To address these limitations, Long Short-Term Memory (LSTM) networks were developed. They incorporate memory cells and gates to control the flow of information, which enable for retaining long-term dependencies mitigating vanishing and exploding gradient problems [18]. BiLSTM is a further development of LSTM, combining the forward hidden layer and the backward hidden layer, which can access both the preceding and succeeding contexts. In contrast, standard LSTMs focus solely on historical context. Therefore, BiLSTMs are better suited for solving sequential modeling tasks than LSTMs [19].

Preprocessing and Embedding for BiLSTM Input

The input preparation process for the BiLSTM model follows the same steps described in CNN Section 3.3. This is because both models require tokenized sequences and numerical embeddings as their inputs. Since embeddings like FastText are designed to represent text data independently of the model's structure, they can easily be applied to preprocessing pipelines for both CNNs and RNNs, including BiLSTMs.

BiLSTM Model Creation

The BiLSTM model implemented for this study is a Bidirectional Long Short-Term Memory network, tailored specifically for sentence classification tasks. Its architecture begins with an embedding layer that converts tokenized input sequences into dense vector rep-

resentations. This layer is initialized with pretrained embeddings (FastText), which are frozen during training to leverage their high-quality representations without altering them.

Following the embedding layer, the model incorporates a Bidirectional LSTM (BiLSTM) layer. This component allows the model to capture both forward and backward dependencies in the input sequence. By combining these contextual features, the BiLSTM layer enables a more comprehensive understanding of the sequential data compared to a unidirectional LSTM.

To reduce dimensionality and focus on the most salient features, the BiLSTM outputs are passed through a global max-pooling layer, which extracts the maximum value for each feature across the sequence. This operation effectively condenses the variable-length sequence data into a fixed-size vector.

Next, a fully connected layer maps the pooled features to the output classes. A dropout layer with a rate of 0.7 is added before this step to prevent overfitting during training by randomly zeroing out a portion of the neurons. Finally, the model outputs logits for each class, which are used to compute the loss and make predictions.

For optimization, we use the Adam optimizer with a learning rate of 1×10^{-3}, a widely used adaptive learning rate optimization algorithm. The model is trained over 20 epochs, with early stopping implemented to halt training if the validation loss does not improve for 5 consecutive epochs. This ensures efficient training while avoiding overfitting.

3.5 BERT (Bidirectional Encoder Representations from Transformers) Transformer

The Transformer model, first introduced in the 2017 paper "Attention is All You Need" [20] by researchers at Google, revolutionized the field of NLP. It is based on Attention mechanism, a technique that can be used by computers as a resource allocation scheme, which is the main means to solve the problem of information overload. By doing that, machines can process most important information focusing on what is crucial and save computing resources as a result [21].

The Transformer architecture is composed of two main parts: the Encoder and the Decoder. Both parts use stacked self-attention mechanisms and fully connected layers. Here is an overview of these components:

- **Encoder**: The encoder takes an input sequence of symbols (e.g., $x_1, x_2, \ldots, x_n$) and maps it to a sequence of continuous representations $(z_1, z_2, \ldots, z_n)$.
- **Decoder**: The decoder takes the encoder's output sequence z and generates the corresponding output sequence $(y_1, y_2, \ldots, y_m)$ one element at a time. At each decoding step, the model is auto-regressive, meaning it consumes the previously generated symbols as additional input when generating the next symbol.

In the Transformer, both the encoder and decoder consist of stacked layers of self-attention mechanism and fully connected feed-forward networks. The encoder processes the input sequence in parallel and outputs a sequence of contextualized representations. The decoder, on the other hand, generates the output sequence step by step, taking into account the representations from the encoder and its own previously generated outputs. The architecture is visualized in Figure 3.2.

BERT (Bidirectional Encoder Representations from Transformers) leverages the power of the Transformer architecture to pre-train contextualized word representations. Unlike traditional unidirectional models, BERT employs a bidirectional approach, capturing both the left and right contexts of words in a sentence during pre-training. This is achieved using the Masked Language Model (MLM) objective, where words are randomly masked, and the model is trained to predict these words based on their context [23]. Such an approach results in a robust understanding of language that can be fine-tuned for a variety of downstream tasks. The architecture is visualized in Figure 3.3.

For this study, we use the pre-trained BERT model provided by Hugging Face, specifically the `bert-base-uncased` variant. The model and its tokenizer are initialized as follows:

```
pretrained_model_name = "bert-base-uncased"
model = TFAutoModel.from_pretrained(pretrained_model_name)
tokenizer = AutoTokenizer.from_pretrained(pretrained_model_name)
```

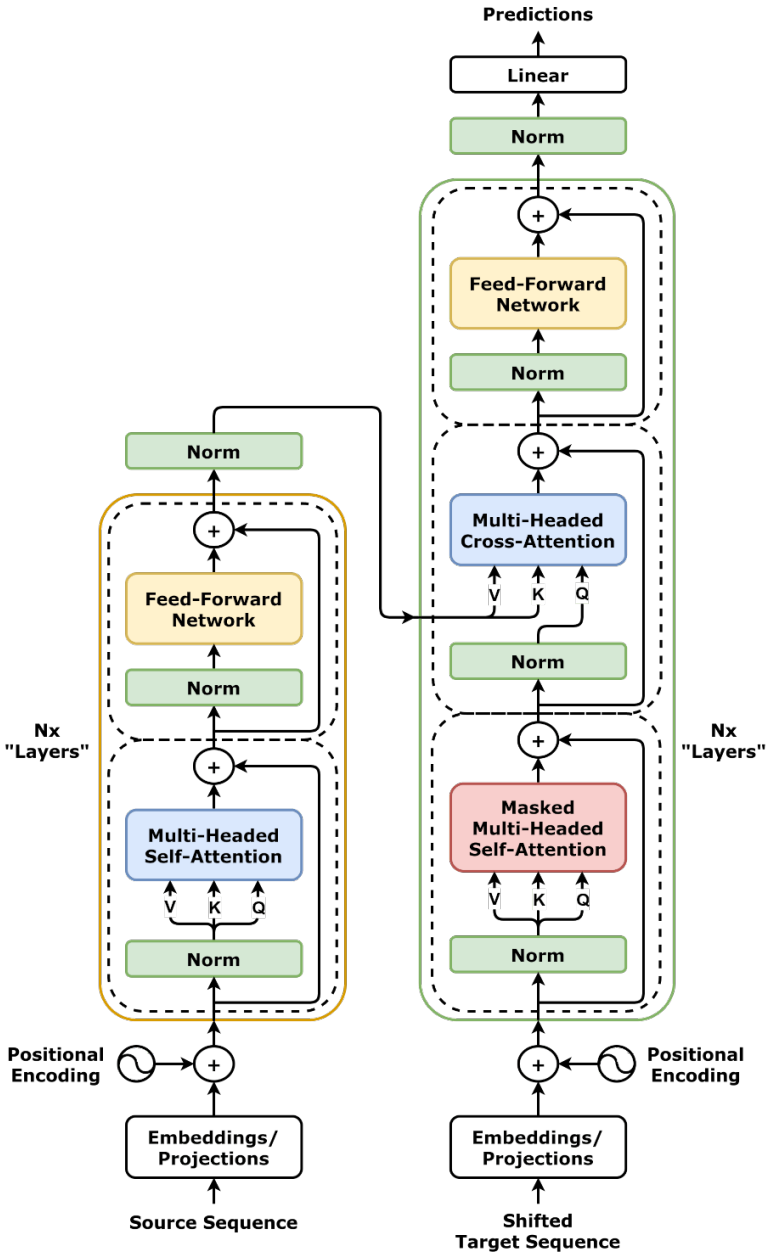


Figure 3.2: Transformer architecture. Image source: Godoy, D. V. (2021). *Illustrations for the Transformer, and attention mechanism. Transformer, full architecture* [22].

After initializing the model and tokenizer, we perform fine-tuning. Fine-tuning involves adapting the pre-trained model to a specific task – in this case, text classification. During fine-tuning, BERT's weights are updated while training on the target dataset, allowing the model to learn task-specific patterns and representations.

Following preprocessing, texts are converted into Hugging Face datasets. Tokenization is applied using the Hugging Face `AutoTokenizer`, ensuring compatibility with the BERT model.

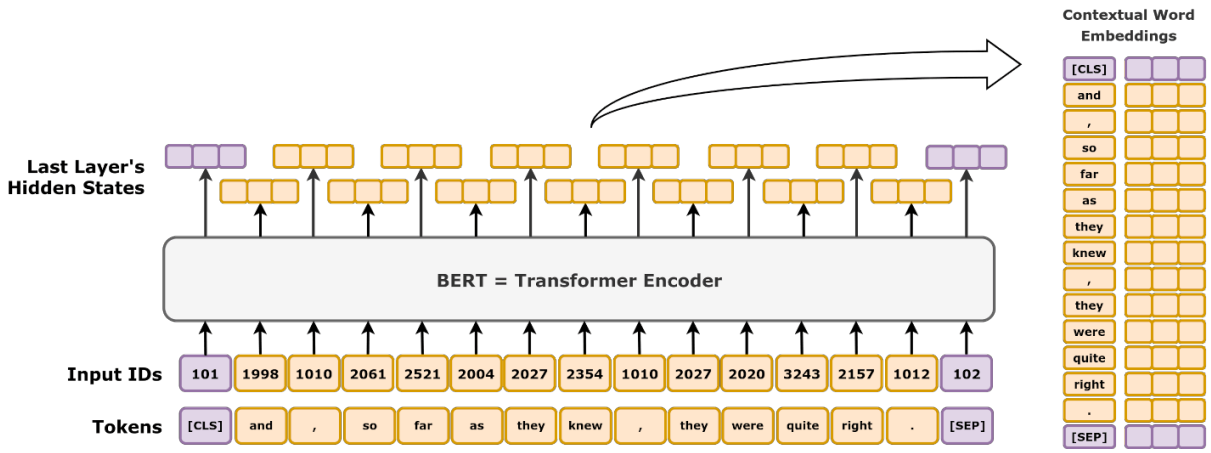


Figure 3.3: BERT architecture. Image source: Godoy, D. V. (2021). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* [24].

The BERT model architecture for classification, which was used in our experiments, is summarized in Table 3.5. The model consists of a pre-trained BERT backbone, a dropout layer, and a classifier layer that performs binary classification:

Component	**Description**
Model	Pre-trained BERT (bert-base-uncased)
Dropout	Dropout layer with rate 0.5
Classifier Layer	Dense layer with 2 classes and softmax activation
Input	Tokenized input data (text)
Output	Predicted class probabilities

Table 3.5: BERT Model Architecture for Classification

While BERT is highly effective, it is computationally expensive. To meet the resource requirements, we utilize Google Colab Pro's most powerful engine, equipped with an NVIDIA A100 GPU (40 GB of VRAM) and 83.5 GB of system RAM. However, due to these constraints, the BERTForClassification model is relatively simple. The model is trained on the entire dataset, but due to the GPU RAM constraint, only 32 instances can be processed in each batch. This results in a total of 3166 batches. Even with a relatively high dropout rate of 0.5, the model converges quickly, allowing for just 2 epochs.

4 Comparative Analysis

With all models creation completed, this chapter focuses on their comparative analysis to highlight performance across different dimensions. This evaluation is structured into five distinct sections, each focusing on a critical aspect of the models' behavior.

4.1 Training Efficiency and Computational Costs

In the context of training our classifiers on our extensive dataset, one of the most crucial differences is training complexity and computational cost. In order to reach the best metrics, it is required to use as many training entries as possible and also fine-tune large space of hyperparameters. This, in itself, makes the process very challenging, even with strong Google Colaboratory resources. These challenges are particularly important when training deep learning models (CNN, BiLSTM, BERT).

In contrast, classic machine learning models like Logistic Regression and SVM have simpler architectures and consequently are less resource-intensive. This simplicity is reflected in the size of the final models: the Logistic Regression model is a lightweight 4.4 MB, while the CNN model occupies over 717 MB. Significantly larger size of deep learning models underscores their ability to capture complex patterns but also highlights challenges in storage and deployment.

GPU acceleration is essential for training deep learning models efficiently. Unlike CPUs, most commonly limited up to 20 cores, GPUs can perform thousands of parallel operations, dramatically reducing training time. Classic models relied on local CPU training, as they did not fully benefit from GPU capabilities. This made them faster to train on a CPU, but less suitable for leveraging the advantages of GPUs, especially with larger datasets.

In summary, GPU usage is essential for efficiently training modern neural networks. In machine learning, the ability to leverage GPU computation is a critical factor. Without GPUs, training models such as BERT would either take an impractical amount of time or require computational power of supercomputers. This reliance highlights critical role of GPUs in advancing AI research, enabling scientists to build more sophisticated models at larger scale.

4.2 Text Representations

Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) are foundational text representation methods that identify textual similarity based on shared vocabulary. However, these approaches do not account for word order, semantic relationships, or word location, which are crucial for deep textual understanding. For instance, sentences like "I love dogs" and "Dogs love me" would generate almost identical vector representations in a BoW model, despite their different meanings.

While TF-IDF shares this limitation, it adjusts word frequencies within a document to reflect how often a term appears relative to the document's length. Additionally, it assigns weights based on the term's rarity across the dataset, giving higher importance to words that are specific to a subset of documents. This dual adjustment makes TF-IDF more effective for ranking terms by relevance in tasks like information retrieval.

Even though BoW and TF-IDF are effective for basic text representation and analysis, their highlighted limitations leave room for more advanced approaches. Modern natural language processing tasks increasingly rely on word embeddings, which address these issues by capturing relationships between words in a continuous vector space. This advancement represents a important shift from frequency-based representations to context-aware vectorization techniques.

Word embeddings, such as those generated by FastText, are a powerful approach to capturing the semantic relationships between words, sentences, or entire documents. Unlike traditional text representation methods, such as BoW and TF-IDF that treat words as discrete, independent entities, word embeddings represent words as dense vectors in a

continuous vector space. These vectors encode rich semantic and syntactic information, allowing for better relationship retrieval.

A classic example of this can be seen in Figure 4.1, which illustrates the analogy "king is to queen as man is to woman." In the figure, words like "king", "queen", "man" and "woman" are represented as vectors in a continuous vector space. These vectors are not just positioned close to each other due to semantic similarity but also maintain parallel relationships, showing their analogous meaning.

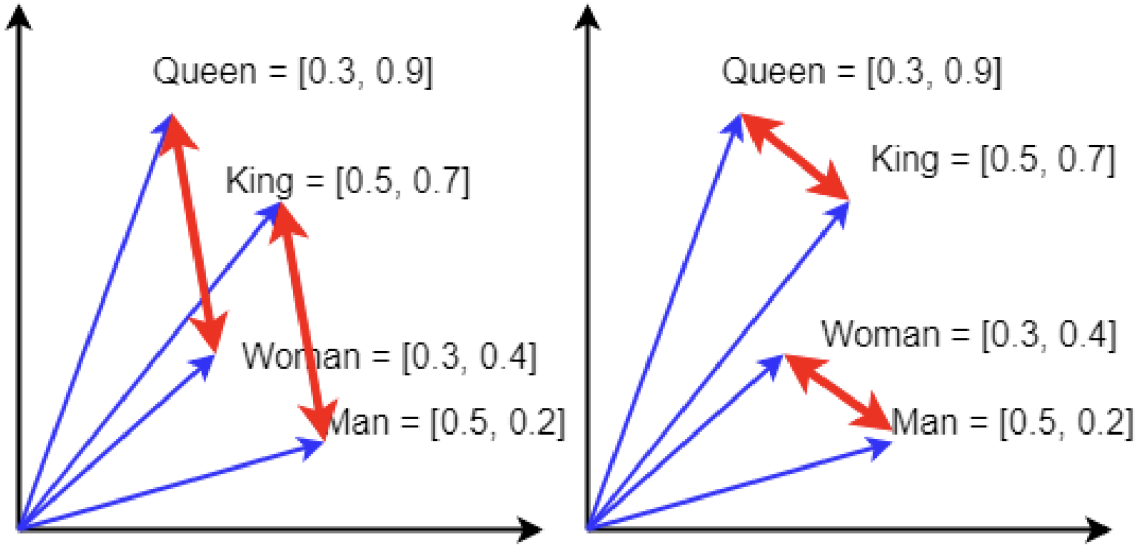


Figure 4.1: Visualization of word embeddings, including the analogy "king is to queen as man is to woman" There are 4 analogies one can construct, based on the parallel red arrows and their direction. [25].

For instance, the vector difference between "king" and "man" is approximately the same as the vector difference between "queen" and "woman." This relational similarity is indicated by the parallel red arrows in the figure, demonstrating that the analogy "king + woman - man = queen" holds true. This kind of relationships enable word embeddings to detect analogy that traditional models like BoW or TF-IDF cannot perform. Even if the vectors are not perfectly aligned, the embeddings are designed to reflect the closest and most meaningful relationships in the data.

This ability to encode complex relationships in a high-dimensional space highlights the strength of word embeddings in capturing nuances and contextual meanings in text, making them much more effective than traditional methods.

In our case, word embeddings generated by fastText significantly enhanced the performance of BiLSTM and CNN models by providing a more meaningful representation of words that is aware of context in the text. Unlike BoW or TF-IDF, embeddings are capable of capturing contextual nuances, such as the difference in meaning between "bank" in "river bank" and "bank" in "financial institution", depending on the surrounding words.

Additionally, FastText stands out among word embedding techniques by modeling both entire words and their subword structures. Unlike Word2Vec and GloVe, which fail to handle out-of-vocabulary (OOV) words, FastText generates embeddings using character-level n-grams. This allows to approximate representations for unseen words. This capability is more robust and ensures greater generalization, making it particularly effective for datasets with diverse vocabularies.

Lastly, BERT uses contextualized word embeddings that address the limitations of earlier methods. Unlike static embeddings, where a word always has the same single vector, BERT dynamically generates embeddings based on the surrounding words. For example, the word "bank" would have two different embeddings in "river bank" and "financial bank", making classification less reliant on further model architecture. BERT's bidirectional approach further enhances its understanding by simultaneously considering both preceding and succeeding words.

Moreover, the fine-tuning capability of BERT allows it to excel in a wide variety of tasks, from sentiment analysis and question answering to classification. It represents the current state-of-the-art for text representation in NLP, setting new benchmarks for performance. However, these advantages come at a certain cost: BERT is significantly more computationally intensive than simpler methods like FastText or TF-IDF. In our case, even the considerable computational resources of Google Colaboratory were insufficient for extensive experimentation with its architecture. Thus, while BERT offers superior performance, selecting a text representation method should carefully take into account these trade-offs.

4.3 Model Interpretations and Context Extraction

The representation of words through methods like TF-IDF and Bag-of-Words is particularly effective for Logistic Regression and SVM models when there are big disparities in word frequency. These methods leverage linear and hyperplane-based decision-making processes of these models.

To analyze this, we compute the frequency distribution of words across human and AI-generated text subsets, normalizing for differences in the average number of words per text. From these distributions, we extracted the 20 most frequent words from the human subset and matched their corresponding frequencies in the AI subset.

Figure 4.2 illustrates these findings. Most frequent words contribute a lot to the overall text composition, with their cumulative frequency summing to 27.29% in human texts and 25.40% in AI texts. Each AI bar is annotated with the relative frequency of the word compared to the human subset.

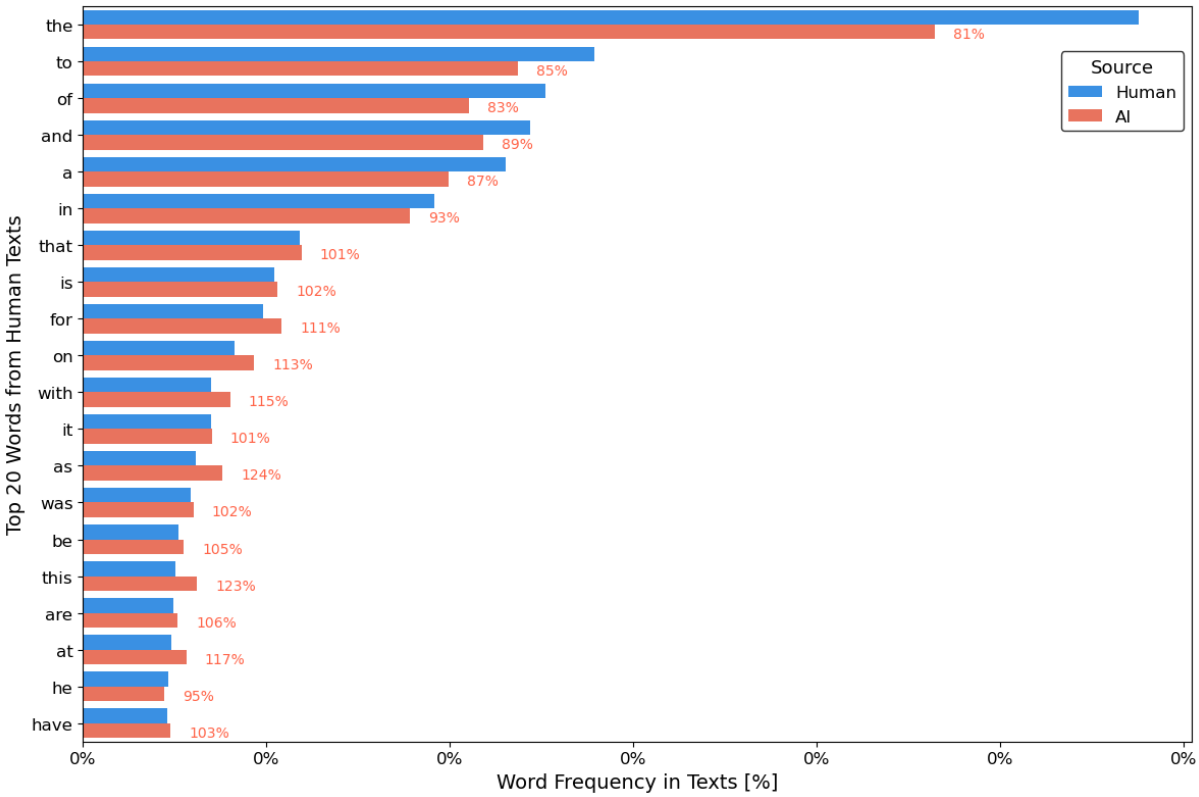


Figure 4.2: Comparison of Word Frequencies in Human and AI-Generated Texts.

Among the 20 most popular words, the relative frequency in AI text ranges from 81% to 124% compared to their usage in human text. For example, the word "the" appears 81% as often in AI text compared to human text. On the other hand, the word "as" is 124% as frequent in AI text.

Apart from the most frequent words, some words are relatively overused in one subset compared to the other. These words are less frequent overall but still reveal stylistic patterns that can be utilized for classification.

Terms like "advertisement", "edit", and "percent" appear significantly more often in human-written texts, as shown in Figure 4.3. Also, humans use contractions like "that's" and "he's" (due to punctuation removal they are displayed as "thats" and "hes" respectively). AI models tend to avoid such contractions, using a more formal style instead.

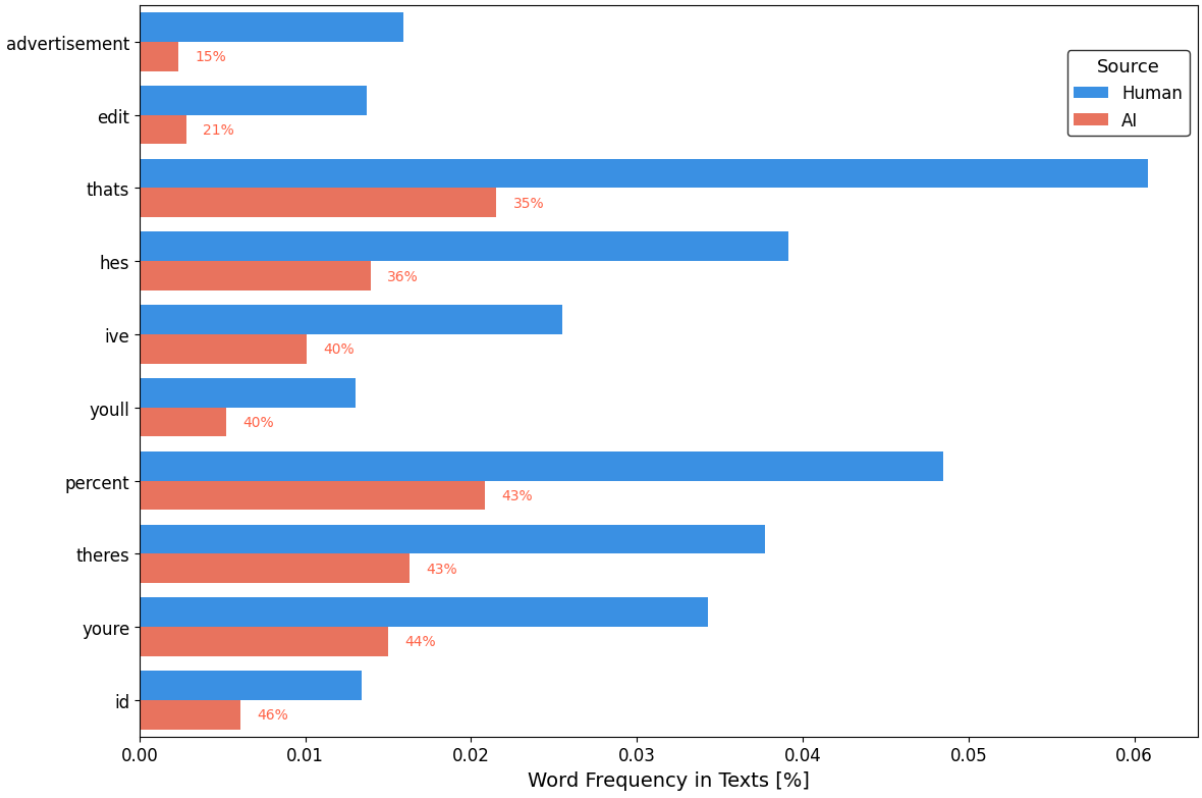


Figure 4.3: Top 10 Words with Highest Relative Usage in Human Texts Compared to AI-Generated Texts.

On the other hand, AI-generated texts use words like "author", "stated", and "due" more often, which can be seen on Figure 4.4. The full list shows a preference for formal and scientific language, which is typical of AI-generated content.

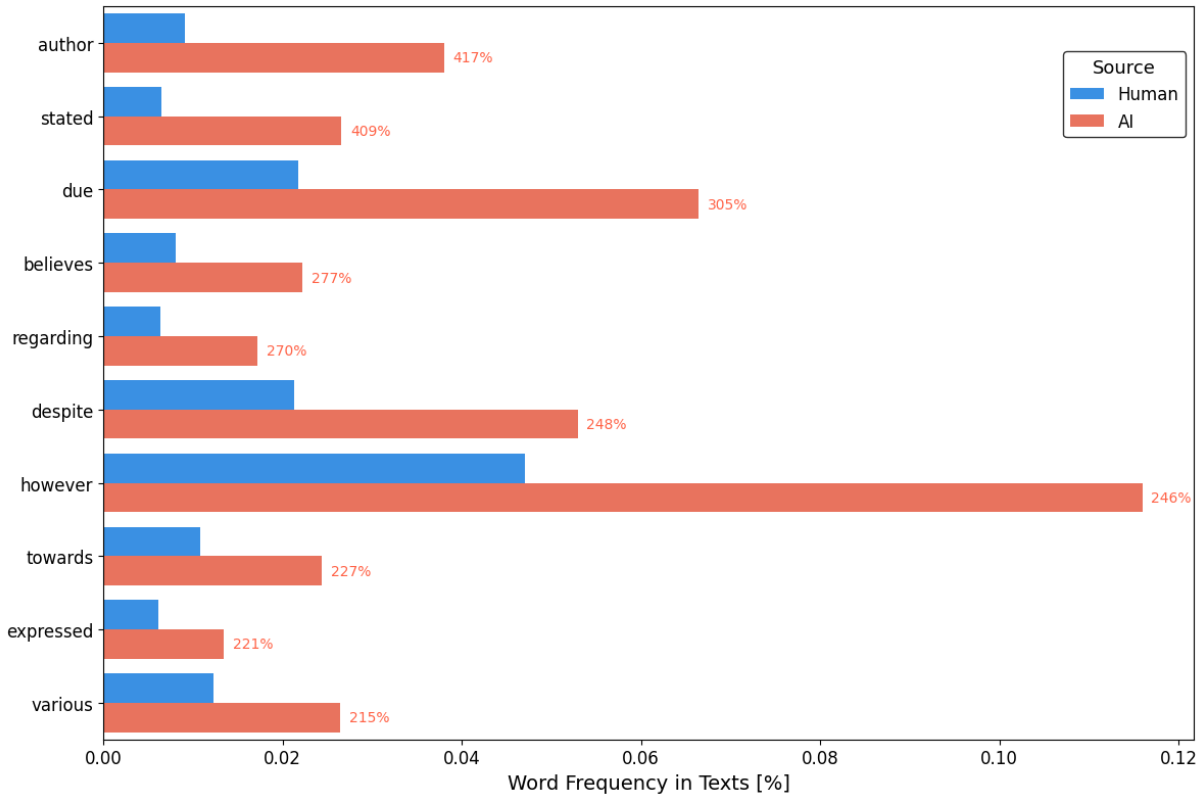


Figure 4.4: Top 10 Words with Highest Relative Usage in AI-Generated Texts Compared to Human Texts.

These differences, while being significant, do not directly translate into exceptional performance metrics. Modern AI-generated text is produced by state-of-the-art LLMs. Unlike deterministic generative methods like greedy or beam search, which might overuse most common words such as "the", "a", "and", the LLMs that generated text in our dataset, use stochastic techniques like top-k and top-p sampling. They ensure more varied generated text, reducing over-reliance on frequent words.

In comparison, more sophisticated models such as CNN and BiLSTM can also leverage occurrences of such words, but do this in a slightly different manner. Both methods use FastText word embeddings, where each word is represented as the same vector. CNN and BiLSTM are capable of extracting information for classification not only from individual word embeddings but also from local combinations of words. However, capturing relationships in longer sequences can be challenging, as these patterns may be rare, even in large datasets.

In CNNs, convolutional layers with filters of varying sizes (from 2 to 5) are applied, as detailed in the CNN model creation section (Section 3.3). These filters capture patterns across sentences of 2 to 5 consecutive words, effectively learning n-gram-like features.

For instance, not frequently used nouns in the dataset may have similar or less meaningful word embeddings. Such nouns are often preceded by determiners like "the", "a", or "an", forming bigrams that certain filters in the convolutional layer can capture. Similarly, filters of length 3 to 5 can identify longer patterns or phrases, such as "in other words", "however, while we can", or "as an AI language model". Despite their strength in capturing local structures, created CNN may struggle with long-range dependencies, as its architecture is focused on local features only.

BiLSTMs overcome this limitation, as they process sequences bidirectionally, extracting insights from both preceding and succeding word contexts. Relationship between words can be extracted in broader context. For instance, BiLSTMs can spot differences in meaning for polysemous words like "bank" in "river bank" versus "financial bank" based on the surrounding words. Despite their ability to handle sequential data, BiLSTMs rely on the quality of the input embeddings to generalize nuanced meanings.

Both CNN and BiLSTM architectures benefit from the use of FastText embeddings, which enhance their ability to represent individual words and subword structures. These embeddings provide clear representations that improve the models' performance, particularly for handling rare words or out-of-vocabulary terms. However, the extraction mechanisms of CNN and BiLSTM operate independently of the embeddings themselves, with FastText only being enhancement rather than the core mechanism of information extraction.

In contrast, our BERT takes a fundamentally different approach to context extraction. Instead of relying on complex defined architectures like CNN or BiLSTM, BERT uses dynamic, context-sensitive embeddings that adapt to the surrounding words. This enables BERT to extract information at a deeper level with more flexibility. For instance, the meaning of "bank" is dynamically determined based on its usage in "river bank" or "financial bank", without requiring additional architecture complexity. In this case, the extraction process is driven by the Transformer layers, which detect relationships across

the entire text, capturing both local and global dependencies. Then, it is wrapped by a simple neural network layer that doesn't add much complexity.

4.4 General Results

Throughout the training of all models, the F1-score is selected as the primary evaluation metric. The F1-score, which represents the harmonic mean of precision and recall, offers a more nuanced assessment of incorrectly classified cases compared to the accuracy metric. This is particularly important in real-world classification problems where imbalanced class distributions are common, making F1-score a more reliable measure of a model's performance.

The F1-score is calculated as follows:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

In our specific case, class imbalance is not a significant issue since the dataset is perfectly balanced, consisting of an equal number of human-written paragraphs and AI-rephrased paragraphs. However, the F1-score remains the preferred metric as it ensures balanced evaluation of both false positives (FP) and false negatives (FN). This balance is critical, as we aim to avoid significant disparities in distinguishing between these two types of errors.

The F1-Score, as well as accuracy, precision and recall can be observed in Table 4.1.

Model	**F1 Score**	**Accuracy**	**Precision**	**Recall**
Logistic Regression	0.887	0.885	0.872	0.904
SVM	0.862	0.864	0.878	0.846
CNN	0.904	0.902	0.885	0.924
BiLSTM (RNN)	0.934	0.935	**0.954**	0.914
BERT	**0.963**	**0.963**	0.952	**0.974**

Table 4.1: Performance Metrics

The evaluation of all five models reveals significant differences in performance across all measured metrics. Among all, the fine-tuned BERT demonstrates superior performance,

achieving the highest results in F1 Score (96.3%), Accuracy (96.3%), and Recall (97.4%). Especially, it presents extremely high recall score, making it even more useful in scenarios where the primary objective is the detect nearly all AI-generated texts. For instance, this would be critically important in applications detecting AI-generated texts in sensitive domains such as news articles or academic submissions, where false negatives could have severe implications. In case of positive evaluation, particular work could be analyzed more deeply with a help of human specialist.

Second best metrics are achieved by BiLSTM (RNN) model, with F1 Score of 93.4%, Accuracy of 93.5%, and the highest Precision (95.4%). This model is particularly effective at minimizing the number of false positives in the dataset, however it comes with the expense of a higher false-negative rate. This performance contrasts with that of BERT, as BiLSTM is particularly effective when avoiding the misclassification of human-written texts as AI-generated is a priority. For example, in cases like filtering emails for spam, where marking an important human-written message as spam could lead to it being lost, BiLSTM's focus on precision makes it a better choice. While it does not perform as well overall as BERT, its ability to minimize false positives makes it valuable in situations where avoiding errors is more important than catching every AI-generated text.

It is important noting that in such specific cases, these models could be further optimized by training them with a customized loss function that assigns greater penalties to false negatives or false positives, depending on the context. Given that BiLSTM's precision is just 0.2% higher than the one in BERT, the later remains preferable choice.

With a similar basis to the BiLSTM model, including the use of FastText word embeddings, the CNN model also demonstrated rather strong performance, achieving an F1 Score of 90.4% and an Accuracy of 90.2%. However, it falls significantly behind BiLSTM by over 3% in these metrics.

Traditional machine learning models such as Logistic Regression and SVM achieved the lowest performance among the tested models. Logistic Regression achieved an F1 Score of 88.7% and Accuracy of 88.5%, while SVM obtained an F1 Score of 86.2% and Accuracy of 86.4%. Nevertheless, these results are still impressive, falling only slightly behind the

more complex deep learning model, CNN. Despite their inability to extract contextual information, these models leverage statistical features to make fairly accurate predictions. They might be particularly useful for creating lightweight models in a fast and simple way.

An interesting observation is that SVM demonstrates an F1 Score 2.5% lower than Logistic Regression, even though its process of model creation is generally considered more advanced. This highlights that increased model complexity does not always translate to better performance. In some cases, simpler models can outperform more complex ones, particularly when they are well-suited to the problem.

4.5 Results Based on Text Length

As highlighted in the data exploration section (Section 2.1), input data for investigated labels varied and it is important to detect whether analyzed models worked correctly for varying text length. In order to do this, we group the test dataset into bins with range of 100 words and we compute accuracy on these subset achieved by each model. Accuracy is chosen here, because F1-score does not have this big utility here because most of the bins have big class imbalanced. The results are displayed in Figure 4.5.

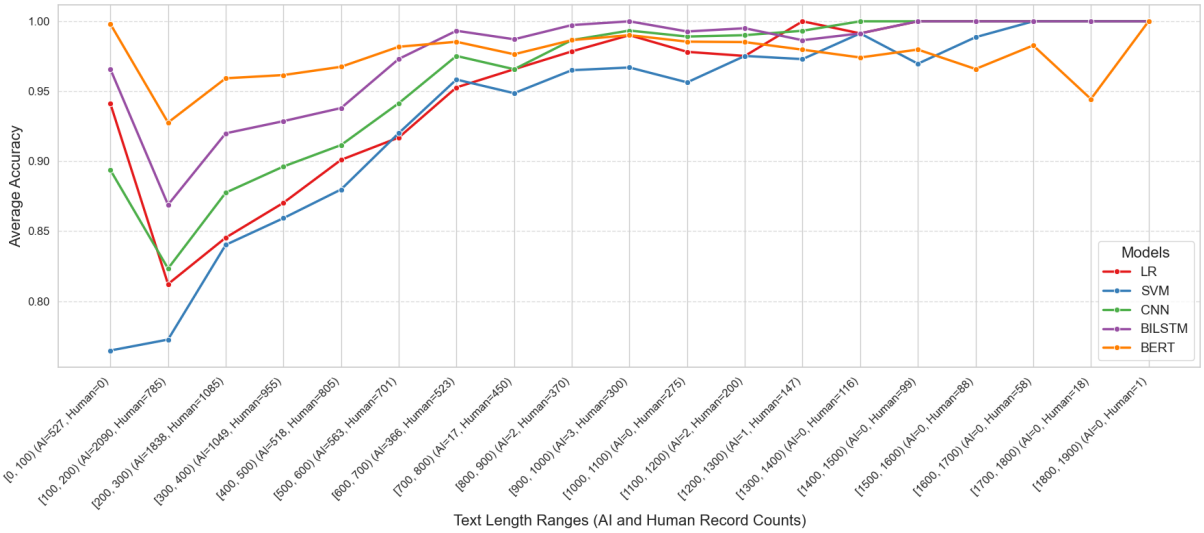


Figure 4.5: Average Accuracy of Models Across Token Length Bins

The analyzed models exhibit similar accuracy trends across most of text length ranges. Generally, model accuracy improves as text length increases, probably because longer texts provide more context and features for classification. For entries exceeding 600 words,

all models achieve an accuracy of over 95%. However, for simpler models like Logistic Regression and Support Vector Machines, this high accuracy is likely influenced by class imbalance, as these models do not effectively capture contextual information. Interestingly, in certain bins, these simpler models outperform more advanced ones, including BERT.

For the shortest texts, particularly those under 100 words, most models perform well, except for SVM, which struggles in this range. These strong performances are likely caused by the fact that the first bin only contains AI-generated entries, allowing models to leverage text length as a key feature.

In contrast, the bins containing texts between 100 and 300 words show the weakest performance. This range includes both AI- and human-generated entries, making it difficult for models to distinguish between sources based solely on length. Additionally, the shorter input length limits the models' ability to perform meaningful syntactic or contextual analysis, further reducing accuracy.

Lastly, it is worth noting that BERT's accuracy slightly decreases for very long texts. This suggests that BERT does not depend heavily on text length, which makes it more robust when handling long inputs, even if the training dataset is slightly biased. This resilience can be advantageous for real-world use-cases involving diverse text lengths.

5 Explainable Artificial Intelligence (XAI)

Machine learning models are becoming increasingly complex, powerful, and able to make accurate predictions. However, as complexity is increasing, they become less interpretable, turning into "black boxes". Understanding how they arrive at their predictions becomes a significant challenge. This has led to a growing focus on machine learning interpretability and explainability, where Explainable Artificial Intelligence (XAI) techniques play a crucial role.

An important aspect of XAI involves in differentiating between two related definitions of trust: (1) trusting a prediction, i.e., whether a user trusts an individual prediction sufficiently to take some action based on it, and (2) trusting a model, i.e., whether the user trusts a model to behave in reasonable ways if deployed. Both depend on how well a human understands a model's behavior, rather than seeing it as a black box.

5.1 Shapley Additive exPlanations (SHAP)

One of the most promising tools in this domain is Shapley Additive exPlanations values, which measure how much each feature contributes to the model's prediction. SHAP values can help you see which features are most important for the model and how they affect the outcome.

SHAP (SHapley Additive exPlanations) values are a way to explain the output of any machine learning model. It uses a game theoretic approach that measures each player's contribution to the final outcome. In machine learning, each feature is assigned an importance value representing its contribution to the model's output.

On the other hand, while SHAP is a powerful tool for interpretability, it is computationally costly. This raises significant challenges when attempting to apply SHAP to any of our deep learning models. Efforts to use SHAP in this study exceeded the available RAM in Google Colaboratory, even in its best environment with A100 GPU, and when tested with a small batch size of 12 inputs. Our models are trained on 101,284 training samples, with an average of 438 words per sample, resulting in a vocabulary of 547,607 unique words. Additionally, every deep learning model created in this study has a complex structure. Another layer of complexity arises from the text processing pipeline itself – text data is first transformed into word embeddings as representations before being fed into the models, resulting in two interconnected layers for interpretation.

As a result, we limit SHAP analysis to the Logistic Regression model, which outperforms the Support Vector Machine (SVM) in terms of performance metrics. Even for this less complex model, using only a 3,000-sample subset of the test data, the RAM usage rises to 42.4 GB.

For this research, we use SHAP Python library. First we create instance of Explainer class, primary explainer interface for this library, with our Logistic Regression model and the Bag-of-Words (BoW) representation of the training dataset. The 3,000-sample test subset is converted into a dense array (from its original sparse BoW representation) to compute SHAP values. By using `summary_plot` method, we can display most important features (words) in this model, as visible in Figure 5.1.

In the summary plot, the x-axis shows the SHAP value (impact on the model output), where negative values decrease the likelihood of the predicted class (AI-generated text with label 1), while positive values increase it. Features on the y-axis are ranked by their overall importance, measured by the average absolute SHAP value.

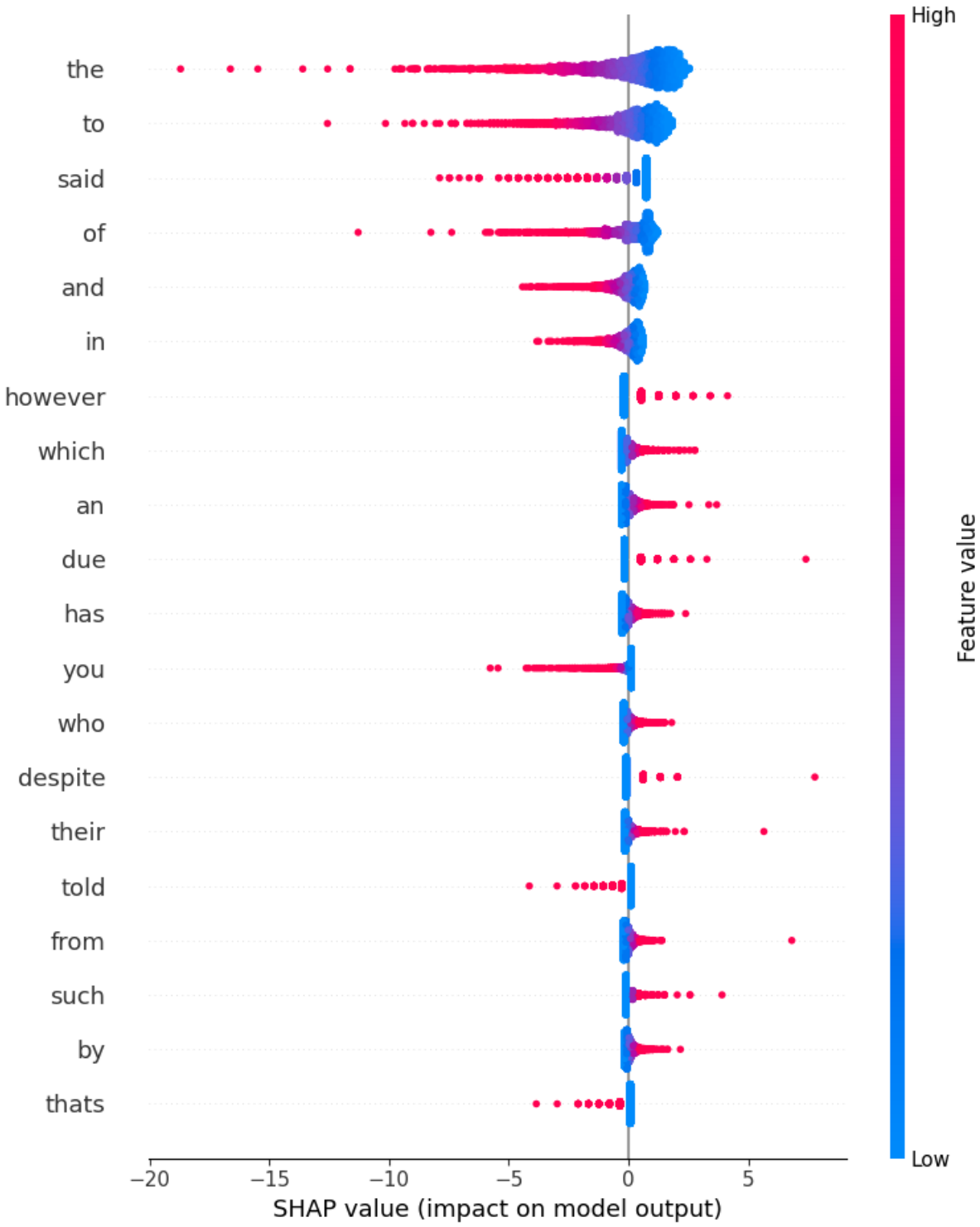
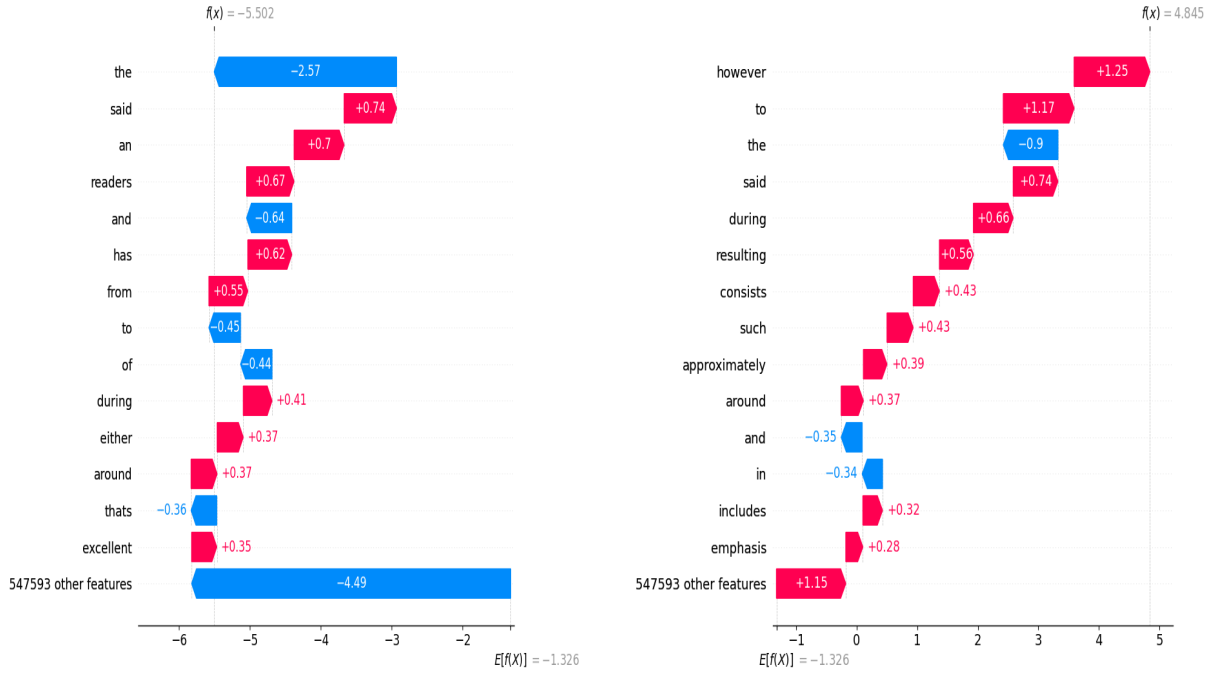


Figure 5.1: Summary plot showing the most important features based on SHAP values for the Logistic Regression model

As observed, these words match significantly with the most popular words in the whole dataset as seen in Figure 4.2. Notably, top two words, "the" and "to" occupy the first two positions in both summaries. However, their effects, like those of most other words visible on the SHAP summary plot, are inconsistent, leading to a wide range of SHAP values.

Words such as "however" and "due" are also impactful, but they also showcase clearer trends; their SHAP values lean consistently in one direction – towards human-written text class.

Next, let us analyze specific inputs from the dataset, as shown in Figure 5.2.



(a) Waterfall plot for correctly predicted human-written text.

(b) Waterfall plot for correctly predicted rephrased text.

Figure 5.2: SHAP waterfall plots showing the feature contributions for the original text (label 0) and its rephrased version (label 1). The plots illustrate how different features (words) impact the model's classification decisions for both texts.

Yet again, words such as "the", "to", "said", and "however" have the biggest impact on the output. We can observe that rephrased text consists of many words categorized as indicative of more advanced language, such as "however", "resulting", "consists", "approximately", "includes", and "emphasis".

Additionally, let us compare the percentages of occurrences of words with the highest SHAP values in these two records against their average occurrences in the entire dataset.

In both examples, the high percentage of occurrences (over 7%) of the word "the" led to a negative SHAP value. As we can see, generally, in human-written texts, 5.7% of the words are "the", while in AI-generated texts, in just 4.64%. Similarly, the word "to" follows the

Word	Avg. Percentile		Percentage in Record	
	Human	**AI**	**True Negative (%)**	**True Positive (%)**
the	5.76%	4.64%	7.09%	7.28%
to	2.79%	2.37%	2.03%	0.88%
said	0.38%	0.20%	0.00%	0.00%
however	0.05%	0.12%	0.00%	0.44%

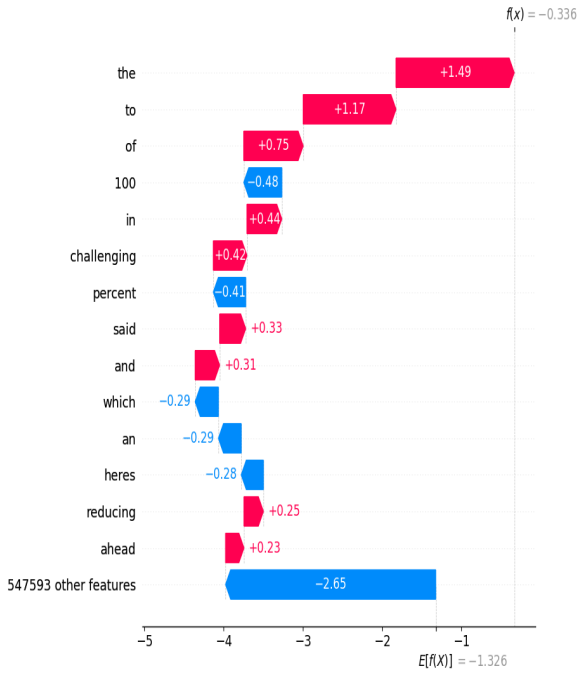
Table 5.1: Comparison of word percentiles in the whole dataset and two analyzed records (True Negative and True Positive).

same pattern: a high percentile of occurrences corresponds to a negative SHAP value. Logistic Regression model assumes that a high usage of these repetitive and easy-to-use words is indicative of human-written text.

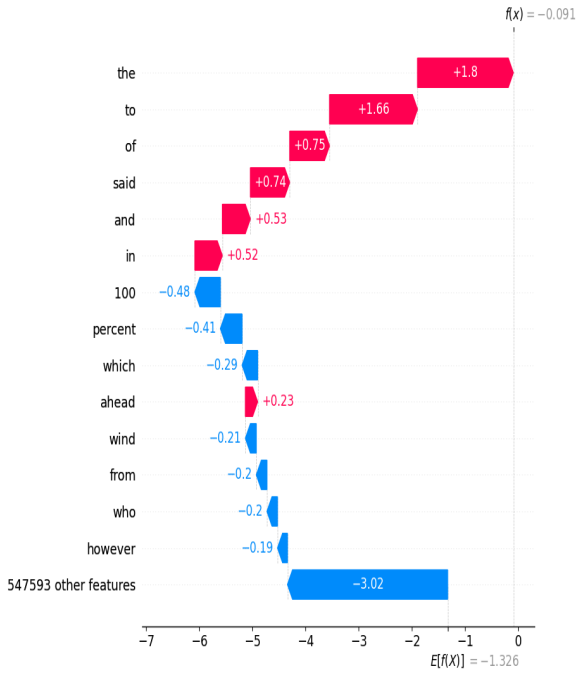
What is interesting, the word "said" does not appear in either of these records, but the model associates the absence of this word with AI-generated text. This word is used 90% more often in human text than in AI-generated text. Also, the word "however" appears approximately 140% more often in AI-generated text, and the model assigns a high SHAP value to it. In the true positive record, "however" was used 0.44% of the time, almost four times as often as in AI texts on average and nearly nine times more than in human-written texts on average. In this case, ChatGPT rephrased the word "but" into this more stylistic word.

This pattern is commonly observed: the use of simple, repetitive words increases the likelihood of the text being written by a human, while the use of more complex, stylistic, and scientific words is associated with AI-generated text.

Let us analyze a pair of texts with a falsely predicted instance, as shown in Figure 5.3.



(a) Waterfall plot for correctly predicted human text.



(b) Waterfall plot for the rephrased text, which was misclassified.

Figure 5.3: Analysis of a text pair: the original human-written text (label 0) is correctly classified, while its rephrased version (label 1) is misclassified. The SHAP waterfall plots highlight the feature contributions for both cases.

This time, the final prediction value $f(X)$ for both the original text and its rephrased version is very similar. We can observe that the SHAP values of certain words are also quite similar. After analyzing this rephrased text, it turns out that around 64% of its length is identical to the original text. LLAMA model did not rephrase it. Only 36% of the AI-generated text differs from the initial input, with 42 words rephrased out of 94 in the human text. This is a common issue observed in LLAMA's text generation, as the model often does not rephrase well-written sections, leading to only a small difference between the original and the generated text.

5.2 Local Interpretable Model-Agnostic Explanations (LIME)

LIME, or Local Interpretable Model-agnostic Explanations, is an algorithm that explains the predictions of any classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction. It was first introduced to machine

learning world in the groundbreaking paper *"Why Should I Trust You?": Explaining the Predictions of Any Classifier* [26]. Researchers were mostly focused on three key aspects:

- **Interpretability**: LIME generates explanations that are easy to understand and provide qualitative insights between the input variables and the response.
- **Local Fidelity**: Explanations are locally faithful, meaning they only explain the model's behavior for the specific instance being predicted, instead of trying to explain how the model works in general.
- **Model-Agnostic**: LIME can be applied to any machine learning model, treating the original model as a black-box. This allows LIME to work with a wide variety of classifiers.

Main LIME idea is that, while the overall behavior of a complex AI model might be difficult to understand, we can explain individual predictions by examining how the model behaves in the vicinity of that prediction.

Additionally, LIME is computationally more efficient than other techniques such as SHAP. Therefore, we can use it on our costly and best performing model – fined-tuned BERT (trained on 20,000 samples).

To achieve this, we utilize the LIME Python library, specifically the LimeTextExplainer class and its `explain_instance` function, which provides explanations for individual predictions. We aim to analyze the same inputs as in Figure 5.2. Let us produce the output of applying this method to a True Negative predicted instance (Figure 5.4) and True Positive instance (Figure 5.5).

Both figures illustrate the prediction probabilities and highlighted word contributions for classifying text as either "Human Text" or "AI Text" using the LIME explainability technique. In a True Negative instance, the model correctly predicts, with a 96% certainty, that the text is human-written. Equivalently, in a True Positive instance, the model makes a correct prediction with 100% certainty.

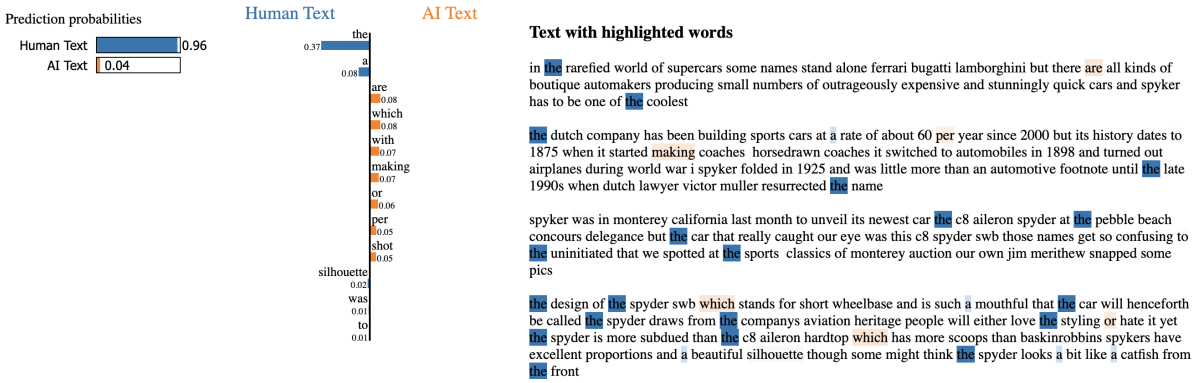


Figure 5.4: LIME explanation for a True Negative predicted instance. The figure highlights the most influential features contributing to the negative classification. Text is truncated.

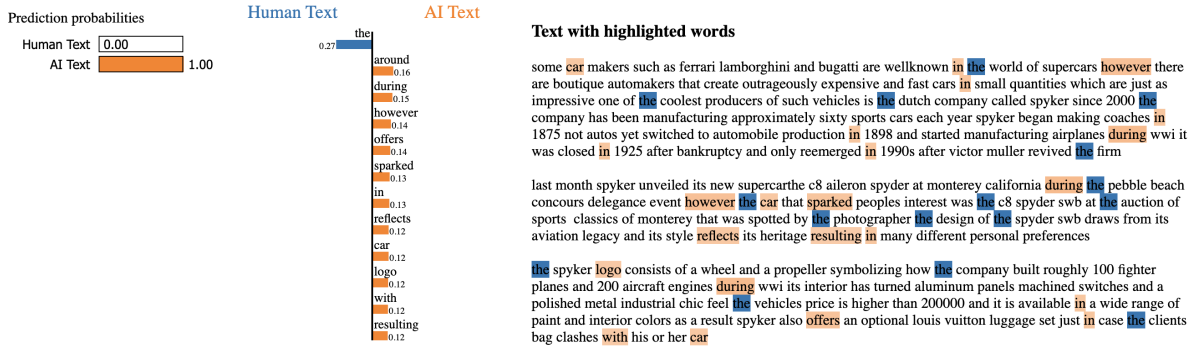


Figure 5.5: LIME explanation for a True Positive predicted instance. The figure shows the features that significantly contributed to the positive classification. Text is truncated.

Similar to the SHAP analysis performed on our Logistic Regression model, the word "the" emerges as the main contributor to the classification result. However, despite very similar percentage occurrences of this word in both texts (as shown in Table 5.1), the feature importance scores assigned by LIME differ. For the True Negative instance, LIME assigns a score of negative 0.37 to "the", whereas, for the True Positive instance, the score is negative 0.27. This disparity highlights BERT's ability to assign different embedding values to the same word based on its surrounding context. This capability underscores the strength of BERT in not only analyzing individual words but also considering their context.

In the True Negative instance, we also observe that, apart from "the", other words present relatively small significance to the final prediction. Among 12 most important features, most exhibit lower positive values – closer to supporting AI Text classification. Interestingly, one of these words is "with" (occurs in a non-visible paragraph). This word appears four times within a single paragraph of 118 words. Such repetition is more characteristic of human-written text, as AI-generated text often uses techniques like top-k and top-p

sampling to avoid overusing a single word in short paragraphs. However, identifying this requires analyzing a much larger portion of the text, a task reliant on nuanced human intuition, something beyond current capabilities of BERT.

In contrast, the True Positive output reveals that, apart from "the", 11 out of the 12 most important features have strong positive values, supporting the classification of the text as AI-generated with high confidence. Consistent with observations from SHAP analysis on the Logistic Regression model, words such as "around", "during", "however", and "resulting" are seen as strong indicators of AI-generated text. These words collectively contribute to a stylistic and complex writing style that is characteristic in AI-produced content. Another notable observation is the word "car". Although used less frequently in the rephrased instance compared to the original, it is assigned a high positive importance value in this case, correctly contributing to the prediction. This once again demonstrates BERT's strength in differentiating predictions based on a word's surrounding context.

6 Concluding Remarks

This research has effectively met its goal of creating, exploring and evaluating various machine learning and deep learning models for distinguishing between human-written and AI-generated text, with a focus on both performance and interpretability.

Among all evaluated models, the fine-tuned, state-of-the-art BERT model performed best across nearly all metrics, achieving an impressive F1-score of 96.3% and maintaining high accuracy across different text lengths. It showed no signs of overfitting and is an adaptable solution for a wide range of applications. Thanks to its superior context extraction capabilities, it is the preferred solution in almost every use case, especially when high accuracy is the most crucial factor.

The BiLSTM model, leveraging FastText embeddings, achieved the second-best results with an F1-score of 93.4% and the highest precision of 95.4%. This makes it particularly effective in minimizing false positives, which is critical in applications where human-written text must not be misclassified. However, its higher false-negative rate limits its effectiveness in scenarios where detecting as many AI-generated texts as possible is crucial.

The CNN model, also based on FastText embeddings, performed well with an F1-score of 90.4% and accuracy of 90.2%, though these metrics were over 3% lower than those of the BiLSTM. While CNNs is effective at capturing local n-grams, they have big limitations in capturing long-range dependencies.

As expected, classic machine learning methods such as Logistic Regression and SVM reached the lowest results. These models, combined with BoW and TF-IDF representations, achieved F1-scores of 88.7% and 86.2%, respectively. Despite their reliance solely

on statistical features and inability to extract context of the statements, they provide a fast and lightweight solution for simpler tasks.

Interpretability analysis revealed that stop words, oftentimes removed in NLP tasks, played a critical role in this particular classification problem. XAI techniques such as SHAP and LIME highlighted the importance of these words, as they generally had the biggest contributions to classification decisions. Another key finding is that rephrased texts consists of many words of more advanced, scientific or stylistic language and models relied on differences of occurrences of such words and their context very much.

This thesis could not fully grasp this problem due to computational constraints that limited the exploration of using advanced neural architectures with BERT transformer. Additionally, the very computationally expensive SHAP analysis, which would have analyzed the whole model, could not be performed on any of the deep learning models, leading to the use of only LIME instance-level analysis on the fine-tuned BERT model.

There are several promising fields for future research:

- **Dataset Expansion:** Using larger and more diverse dataset, including newer AI-generated texts from models like ChatGPT, LLAMA, and Gemini, as well as human-written inputs from various web sources, could improve classification performance. In this study, we focused only on fully rephrased texts, which limits its applicability. Further research could explore datasets containing partially modified texts, as these are more common in real-world scenarios. Moreover, entirely AI-generated texts (created from prompts) should be analyzed to cover all possible input cases.

- **Hyperparameter Tuning:** Experimenting with learning rates, dropout rates, and batch sizes, as well as exploring advanced fine-tuning techniques in all models discussed, might lead to additional gains.

- **Advanced BERT Training:** Extending the training process for BERT with larger batch sizes and more epochs could enhance performance further. Employing BERT in a hybrid architecture, such as combining it with BiLSTM, could improve contextual extraction and classification accuracy.

- **Deeper XAI Integration:** Applying Explainable AI (XAI) to analyze model-level behavior, rather than focusing only on individual predictions, can provide deeper insights into the "black box" nature of complex models like CNNs, BiLSTMs, and fine-tuned BERT. This would deepen our understanding of their decision-making processes.

By achieving its objectives, this study not only advances the classification of AI-generated and human-written text but also provides a generalizable basis for other text classification tasks. The framework developed in this study is flexible and can be easily adjusted to work with other binary or general classification problems.

However, this task will likely become increasingly challenging. As these generative models continue to evolve, their ability to mimic human writing will improve gradually, making classification more complex. Over time, even the most advanced classifiers may struggle to maintain their current levels of accuracy. Furthermore, generative models could potentially be trained to bypass detection by leveraging classifiers like those discussed in this paper. This connection between text generation and classification highlights the need for continuous innovation in both fields.

REFERENCES

[1] R. Pfeifer and C. Scheier, *Understanding intelligence*. MIT press, 2001.

[2] J. McCarthy, "What is artificial intelligence," http://jmc.stanford.edu/articles/whatisai.html, 2007, accessed: 2024-12-19.

[3] M. Hutter, "Universal algorithmic intelligence: A mathematical top→down approach," in *Artificial General Intelligence*, ser. Cognitive Technologies, B. Goertzel and C. Pennachin, Eds. Berlin: Springer, 2007, pp. 227–290, accessed: 2024-12-18. [Online]. Available: http://www.hutter1.net/ai/aixigentle.htm

[4] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier *et al.*, "ChatGPT for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102274, 2023.

[5] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G. Yang, "XAI-explainable artificial intelligence," *Science Robotics*, vol. 4, no. 37, 2019.

[6] W. Neuman, "Ai content detectors," https://github.com/WojciechNeuman/ai-content-detectors, 2025, accessed: 2025-01-23.

[7] T. Borovicka, M. Jirina Jr, P. Kordik, and M. Jirina, "Selecting representative data sets," *Advances in data mining knowledge discovery and applications*, vol. 12, pp. 43–70, 2012.

[8] Y. Chen, H. Kang, Y. Zhai, L. Li, R. Singh, and B. Raj, "OpenLLMText dataset," 2023, creative Commons Attribution 4.0 International (CC-BY 4.0). Accessed: 2024-09-25. [Online]. Available: https://doi.org/10.5281/zenodo.8285326

[9] J. Ramos *et al.*, "Using TF-IDF to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, no. 1. Citeseer, 2003, pp. 29–48.

[10] M. A. Chandra and S. Bedi, "Survey on svm and their application in image classification," *International Journal of Information Technology*, vol. 13, no. 5, pp. 1–11, 2021.

[11] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[12] J. Shin, Y. Kim, S. Yoon, and K. Jung, "Contextual-CNN: A novel architecture capturing unified meaning for sentence classification," in *2018 IEEE international conference on big data and smart computing (BigComp)*. IEEE, 2018, pp. 491–494.

[13] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, October 25–29 2014, pp. 1746–1751. [Online]. Available: https://aclanthology.org/D14-1181.pdf

[14] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," 2016, accessed: 2024-12-22. [Online]. Available: https://arxiv.org/abs/1510.03820

[15] S. Ghannay, B. Favre, Y. Esteve, and N. Camelin, "Word embedding evaluation and combination," in *Proceedings of the tenth international conference on language resources and evaluation (LREC'16)*, 2016, pp. 300–305.

[16] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[17] C. Tran, "CNN for sentence classification," February 2020, accessed: 2024-12-22. [Online]. Available: https://chriskhanhtran.github.io/posts/cnn-sentence-classification

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[19] G. Liu and J. Guo, "Bidirectional LSTM with attention mechanism and convolutional layer for text classification," *Neurocomputing*, vol. 337, pp. 325–338, 2019.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023, accessed: 2024-12-22. [Online]. Available: https://arxiv.org/abs/1706.03762

[21] Z. Niu, G. Zhong, and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, 2021.

[22] D. V. Godoy, "Illustrations for the transformer, and attention mechanism. transformer, full architecture," https://github.com/dvgodoy/dl-visuals/?tab=readme-ov-file, 2021, accessed: 2024-12-27. [Online]. Available: https://github.com/dvgodoy/dl-visuals/?tab=readme-ov-file

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186, accessed: 2025-01-12. [Online]. Available: https://aclanthology.org/N19-1423/

[24] D. V. Godoy, "BERT: Pre-training of deep bidirectional transformers for language understanding," https://dvgodoy.github.io/dl-visuals/BERT/, 2021, accessed: 2024-12-27. [Online]. Available: https://dvgodoy.github.io/dl-visuals/BERT/

[25] P. Sutor, Y. Aloimonos, C. Fermuller, and D. Summers-Stay, "Metaconcepts: isolating context in word embeddings," in *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE, 2019, pp. 544–549.

[26] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?' Explaining the Predictions of Any Classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.