University of Tripoli / Faculty of Science

Department of Computer Science

Evaluating the Effectiveness of Machine Learning Algorithms for Spam Detection

In partial fulfillment of the criteria for the Bachelor of Science (BSc) degree in Computer Science

Prepared by the students

Hadiel Ali Aljadid

2200106086

May Moktar Algallai

2180103961

Supervised by

Dr. Ali Aburas

Spring 2024

Abstract

The ever-growing influx of spam emails poses a significant challenge for email users and security systems. This thesis investigates the effectiveness of Bidirectional Encoder Representations from Transformers (BERT), a powerful pre-trained language model, for spam email detection. BERT's ability to understand complex language context and handle longer sequences of text makes it particularly well-suited for this task. Spam emails often rely on deceptive wording and can be lengthy, requiring models that can analyze these nuances effectively. This research employs BERT on a dataset of labeled spam and ham emails. The emails undergo preprocessing to prepare them for analysis, and relevant features are extracted through feature engineering techniques. Feature engineering involves cleaning the text data and transforming it into a format suitable for BERT's input, such as tokenization, attention masks, and token type IDs. The performance of the BERT model is evaluated using standard metrics like accuracy, precision, and recall. Our findings are expected to demonstrate the effectiveness of BERT in achieving high accuracy for spam email detection. Additionally, this research sheds light on the specific advantages of using BERT for spam filtering compared to traditional machine learning approaches such as Naive Bayes, Support Vector Machines, Decision Trees, Logistic Regression.

Contents

Figures

Chapter 1

1.1 Introduction

The ever-increasing volume of spam emails poses a significant challenge for email users and security systems. These unsolicited commercial emails (UCE), commonly known as spam, disrupt user productivity, consume valuable storage space, and can lead to security breaches. According to a 2023 report by Statista, over 45% of global email traffic is spam, representing billions of emails daily. The financial impact of spam is also substantial, with businesses losing an estimated $20.5 billion annually due to productivity loss, phishing scams, and the costs associated with filtering these emails, (Statista, 2023)**.**

Spam emails can lead to more severe consequences beyond mere annoyance, such as phishing attacks, where fraudulent emails trick users into providing sensitive information, leading to identity theft and financial loss. Notable incidents, such as the 2016 attack on the Democratic National Committee, where phishing emails led to a significant data breach, highlight the potential risks. The costs of such breaches are immense, both in terms of financial loss and damage to reputation, (Times, 2016).

Traditionally, users and email providers have employed various techniques to screen and filter spam emails. Manual filtering involves users flagging unwanted emails, a method that is time-consuming and often ineffective at handling large volumes. Automated filtering, such as rule-based systems (G. V. Cormack, 2007), relies on predefined rules and keyword matching to identify spam. However, these methods have limitations, particularly as spammers develop more sophisticated techniques to bypass these filters. For instance, by deliberately misspelling words or using images instead of text, spammers can evade traditional keyword-based filters, (Sahami M. e., 1998)

Given these challenges, there is a growing need for more advanced techniques to effectively combat spam. Machine learning (ML) has emerged as a powerful tool for spam email detection, offering a dynamic approach that can adapt to new spam tactics. ML algorithms can analyze vast amounts of data, learn from patterns, and improve their accuracy over time. Unlike static rule-based systems, ML models can continuously evolve, making them more robust against evolving spam strategies. (Sculley et al, 2009)

Several ML algorithms have been applied to the task of spam detection, each with its strengths and weaknesses. Support Vector Machines (SVM) are effective at handling high-dimensional data and have been widely used for text classification tasks (Cortes & Vapnik, 1995), including spam detection. Decision trees offer a clear and interpretable model, making them useful for understanding the decision-making process behind spam classification (Quinlan, 1986). Logistic regression, a simple yet powerful algorithm, is often used as a baseline model in spam detection studies (Hosmer et al, 2013). Naive Bayes, known for its simplicity and speed, performs well with smaller datasets and has been a popular choice for spam filtering for many years (Metsis et al, 2006). However, the performance of these traditional algorithms is often limited when dealing with the complexities of natural language.

In recent years, the introduction of Bidirectional Encoder Representations from Transformers (BERT) has revolutionized natural language processing (NLP) tasks, including spam detection. BERT's ability to understand complex language context and handle longer sequences of text makes it particularly well-suited for this task. By leveraging BERT, spam detection systems can achieve higher accuracy rates, even in the face of increasingly sophisticated spam tactics, (Devlin et al., 2019).

In our project, we will investigate the effectiveness of various machine learning algorithms, including Support Vector Machines (SVM), decision trees, logistic regression, naive bayes, and Bidirectional Encoder Representations from Transformers (BERT), for spam email detection. Our focus will be on comparing the performance of these models, particularly highlighting how BERT's advanced language understanding capabilities offer a significant advantage in identifying and filtering out spam emails.

1.2 Spam emails

Spam emails, often referred to simply as "spam," are unsolicited and often irrelevant or inappropriate messages sent over email to a large number of recipients. These emails are typically sent for commercial purposes, such as advertising products, services, or even scams. Spam emails can also be used for phishing, where the sender tries to deceive the recipient into revealing personal information, such as passwords or credit card numbers.

1.3 Importance of Spam Filtering

Effective spam filtering plays a crucial role in safeguarding email users and systems from the various threats posed by spam emails. Here's a closer look at the problems spam emails cause and why robust spam filtering is essential:

- ***Reduced Productivity***: The constant influx of spam emails can overwhelm inboxes, forcing users to sift through irrelevant messages to find legitimate emails. This wastes valuable time and reduces overall productivity, it was noted that an average office worker receives over 120 emails per day, with nearly half being unsolicited. This not only distracts employees but also leads to significant productivity losses across businesses globally. (Radicati, 2020-2024).
- ***Storage Consumption***: Spam emails can consume significant storage space on email servers and user devices. This is a major concern for businesses with limited storage capacity and individuals with limited data plans, reported that organizations could see a 30% increase in storage costs if spam is not adequately filtered. In one notable case, a small business in the UK reported spending an extra £2,000 annually on storage upgrades to handle excessive spam. (Symantec, 2019).
- ***Security Risks***: Spam emails often contain malicious attachments or links that can lead to phishing attacks, malware infections, or other security breaches. Clicking on these links or

opening attachments can compromise sensitive information or damage computer systems, Data Breach Investigations Report highlighted that 94% of malware is delivered via email, underscoring the significant security risks posed by spam. (Verizon, 2020).

- Financial Losses: Some spam emails promote scams or fraudulent activities that can trick users into revealing personal information or sending money. Effective spam filtering helps protect users from falling victim to these scams, which revealed that Business Email Compromise (BEC) scams, often delivered via spam, led to losses exceeding $1.8 billion in that year alone, (FBI., 2020).
- Network Strain: The high volume of spam emails can overload email servers, leading to delays in legitimate email delivery and impacting network performance. Effective spam filtering helps alleviate this strain, reported that during the global pandemic, the volume of spam surged by over 30%, causing widespread network slowdowns and forcing many organizations to upgrade their infrastructure at significant cost. (Cisco., 2020).

1.3.1 Impact on Businesses:

For businesses, the consequences of ineffective spam filtering can be even more severe. Spam emails can:

- Damage Brand Reputation: Spam emails sent from a spoofed company address can damage the organization's reputation and erode customer trust, documented an incident where a multinational corporation's reputation suffered after cybercriminals sent phishing emails impersonating the company, leading to a 20% drop in customer trust scores. (Cisco., 2020).
- Reduce Employee Productivity: As mentioned earlier, dealing with spam emails can significantly impact employee productivity, showed that a mid-sized company in the United States saw a 15% decrease in productivity due to spam, equating to an annual loss of approximately $1.2 million, (Radicati, 2020-2024).
- Increase IT Costs: Businesses need to invest in robust email security solutions and employee training to combat spam effectively, one large enterprise spent an additional $500,000 on

IT security measures in response to a significant spam attack that nearly crippled its email systems, (Symantec, 2019).

1.4 Main Problem

The proliferation of spam emails has become a pressing concern in today's digital world, posing significant challenges for both individuals and businesses. With the constant influx of unsolicited messages, there is a growing need for more effective spam detection systems to protect users from potential security risks and maintain productivity. Traditional machine learning algorithms have long been employed to address this problem, utilizing statistical methods and feature engineering to classify emails as spam or non-spam. However, as the complexity of spam emails evolves, the limitations of these traditional approaches become apparent.

Recent advancements in natural language processing (NLP) have introduced more sophisticated models, such as Bidirectional Encoder Representations from Transformers (BERT), that leverage deep learning techniques to understand the context and semantics of text at a granular level. BERT, with its ability to capture intricate word relationships and context, represents a significant leap forward in NLP and has demonstrated impressive results across various text classification tasks.

Given the potential of BERT, it is imperative to explore how this state-of-the-art model compares to traditional machine learning algorithms specifically in the domain of spam detection. The goal is to determine whether BERT's advanced capabilities translate into tangible improvements in identifying spam emails and to assess whether these improvements justify its computational demands and complexity. Additionally, understanding the strengths and weaknesses of BERT in comparison to algorithms like Logistic Regression, Decision Trees, Support Vector Machines (SVM), and Naive Bayes can provide valuable insights into the practical application of these models in real-world spam filtering systems.

How does the performance of BERT, a state-of-the-art model for spam detection, compare to traditional machine learning algorithms such as Logistic Regression, Decision Trees, Support Vector Machines, and Naive Bayes?.

This research question aims to explore and compare the effectiveness of BERT, a powerful deep learning model, with various traditional machine learning algorithms in the specific task of spam detection. By evaluating the performance of BERT alongside Logistic Regression, Decision Trees, Support Vector Machines (SVM), and Naive Bayes, we seek to understand:

1. **Performance Metrics:** How do these algorithms compare in terms of accuracy, precision, recall, F1-score, and other relevant metrics in detecting spam emails?
2. **Strengths and Weaknesses:** What are the unique strengths and weaknesses of BERT compared to traditional algorithms in the context of spam detection? This includes aspects such as computational efficiency, interpretability, robustness to overfitting, and ability to handle high-dimensional and unstructured data.
3. **Effectiveness of BERT:** Is BERT significantly more effective than traditional machine learning algorithms for spam detection, and in what specific ways does it outperform or fall short compared to these algorithms?
4. **Practical Application:** Considering factors like ease of implementation, scalability, and performance on large and diverse datasets, which approach is most suitable for real-world spam detection systems?

By answering this research question, we aim to provide a comprehensive analysis of the comparative performance of BERT and traditional machine learning algorithms, offering valuable insights into their practical application in spam detection systems.

1.5 Classification Algorithms Used

This project employs **supervised learning**, where the machine learning model is trained on a dataset of emails labeled as either spam or ham. During the training phase, the model learns patterns and features unique to each category from these labeled examples. Once trained, the model applies this knowledge to classify new, unseen emails. Although each classification algorithm used in this project follows the same supervised learning approach, they differ in their internal mechanics and decision-making processes.

The process begins with an incoming email, which is fed into the machine learning model. The model analyzes the content of the email and, based on the training it has received, classifies the email into one of two categories: *Spam* or *Ham*. As shown in **Figure 1**, this workflow is central to the functioning of our spam detection system. While the structure remains consistent, the classification algorithms used in this project vary in terms of their specific techniques for email classification.

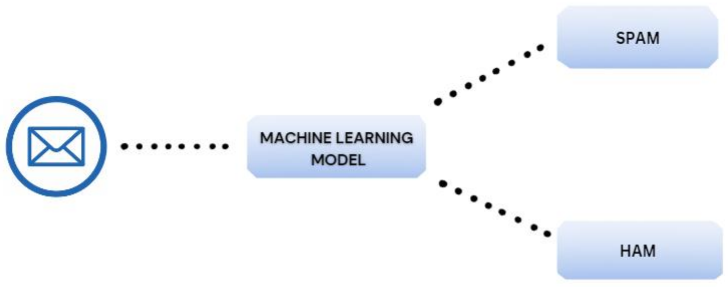


Figure 1 : Spam detector

In our project on spam detection, we employed several popular classification algorithms to compare their effectiveness. These algorithms were chosen based on their proven performance in various text classification tasks, including spam detection. The primary goal was to accurately distinguish between spam and legitimate (ham) emails, minimizing false positives and false negatives.

1.5.1 Logistic Regression (LR)

Logistic regression is a widely used linear model for binary classification tasks. The core idea behind logistic regression is to estimate the probability that a given input belongs to a particular class. It achieves this by fitting a logistic function, also known as the sigmoid function, to the input data. The logistic function is defined as:

$$P(Y = 1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_p X_p)}}$$

where β0,β1,…,βp are the parameters of the model. These parameters are typically estimated using maximum likelihood estimation. The logistic function maps any real-valued number into the (0, 1) interval, making it suitable for modeling probabilities.

One of the key strengths of logistic regression is its simplicity and interpretability. The model's coefficients can provide insights into the relationship between the features and the target variable. For example, in spam detection, the coefficients indicate how much each word contributes to the likelihood of an email being spam. Despite its simplicity, logistic regression performs well across a variety of problems and serves as a robust baseline model. However, it assumes a linear relationship between the features and the log-odds of the outcome, which might not always hold true, (Hosmer et al, 2013).

1.5.2 Decision Trees (DT)

Decision trees are non-linear models that create a tree-like structure of decisions by recursively splitting the data into subsets based on feature values. Each internal node represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label. The splitting process continues until the data at each node are homogeneous or another stopping criterion is met.

The primary advantage of decision trees is their interpretability and ease of visualization. They can handle both numerical and categorical data, making them versatile for various types of classification tasks. The tree structure can be easily visualized, allowing for straightforward interpretation of the model's decision-making process.

However, decision trees are prone to overfitting, especially when the tree grows too deep. Overfitting occurs when the model captures noise in the training data, leading to poor generalization to new data. Techniques such as pruning (removing branches that have little importance) and ensemble methods like bagging and boosting can help mitigate overfitting, (Quinlan, 1986).

1.5.3 Support Vector Machines (SVM)

Support Vector Machines (SVM) are powerful classifiers that aim to find the optimal hyperplane that separates the classes in the feature space. The objective is to maximize the margin, which is the distance between the hyperplane and the nearest points from either class, known as support vectors. The optimization problem can be formulated as:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1, \quad \forall i$$

where w is the weight vector, b is the bias, and yi are the class labels. y_i

SVMs are particularly effective in high-dimensional spaces and are suitable for problems where the number of dimensions exceeds the number of samples. They can handle both linear and non-linear boundaries through the use of kernel functions, such as linear, polynomial, and radial basis function (RBF) kernels. The choice of kernel allows SVMs to map the input space into higher-dimensional spaces where a linear separation is possible.

Despite their effectiveness, SVMs can be computationally intensive, especially with large datasets. However, they are known for their robustness and ability to achieve high accuracy in classification tasks, (Cortes & Vapnik, 1995).

1.5.4 Naive Bayes (NB)

Naive Bayes classifiers are based on Bayes' theorem and assume that the features are conditionally independent given the class. This strong independence assumption simplifies the computation of the posterior probabilities. The probability of a class C given a feature vector X is computed as:

$$P(C|X) = \frac{P(C)\prod_{i=1}^{n} P(X_i|C)}{P(X)}$$

where P(C) is the prior probability of the class, P(Xi|C) is the likelihood of feature Xi given the class, and P(X) is the probability of the feature vector.

Despite the independence assumption often being violated in real-world data, Naive Bayes performs surprisingly well in many classification tasks, particularly in text classification. It is computationally efficient and requires a small amount of training data to estimate the parameters. This efficiency makes it suitable for real-time applications and large datasets.

In the context of spam detection, Naive Bayes is effective due to the large number of features (words) and the assumption that the presence of one word is independent of the presence of another, given the class label (spam or ham). This makes it a popular choice for email filtering and other text-based classification problems, (Metsis et al, 2006).

1.5.5 BERT (Bidirectional Encoder Representations from Transformers)

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art natural language processing (NLP) model developed by Google that utilizes deep learning to understand the context of words in relation to surrounding words. Unlike traditional models that rely on feature extraction techniques like Bag of Words (BoW) or Term Frequency-Inverse Document Frequency (TF-IDF), BERT captures the semantic meaning of text by considering the full context of a word in both directions (left-to-right and right-to-left) using a transformer architecture .

In the context of spam email detection, BERT can be fine-tuned to classify emails as spam or not spam by learning from labeled datasets during its fine-tuning phase. Before discussing how BERT is applied to this task, it’s important to understand its two main phases: pre-training and fine-tuning.

1.5.5.1 Pre-training Phase

BERT's pre-training phase is a critical step where the model learns a general understanding of language. This phase involves training on a large corpus of text data, using two main tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The pre-training phase includes several key components:

1. **Tokenization and Token Embeddings**:

- BERT starts by breaking down the input text into smaller units known as tokens. These tokens can be individual words or subword units, depending on the vocabulary used .
- Each token is then converted into a dense vector representation called a token embedding. These embeddings are numerical representations that capture the syntactic and semantic properties of each token .

- For example, in the sentence "This is a spam email," the tokens might be ["This", "is", "a", "spam", "email"] and each would have a corresponding token embedding that represents its meaning in a high-dimensional space.

2. **Segment Embeddings**:

- To handle inputs that consist of sentence pairs (e.g., question-answer pairs or sequences of sentences), BERT includes segment embeddings.
- Each token is assigned a segment embedding that indicates whether it belongs to the first or the second sentence in a pair. This helps the model differentiate between different parts of the input and understand relationships between sentences.
- For instance, if analyzing an email's subject and body separately, segment embeddings would identify which tokens belong to the subject and which to the body.

3. **Position Embeddings**:

- Unlike recurrent neural networks (RNNs), transformers do not inherently capture the order of tokens. Therefore, BERT introduces position embeddings to encode the position of each token within the sequence.
- Position embeddings are added to the token embeddings to ensure that the model understands the order of words in the sentence .
- This is crucial in email detection because the order of words can change the meaning of a sentence, and thus its classification as spam or not spam.

4. **Masked Language Modeling (MLM)**:

- During pre-training, BERT randomly masks some of the tokens in the input sequence and then attempts to predict the original tokens based on the context provided by the other, unmasked tokens .

- This task forces BERT to develop a deep understanding of context and word relationships, which is essential for tasks like spam detection where the model needs to understand subtle nuances in language.

5. **Next Sentence Prediction (NSP)**:

- BERT is also trained to predict whether one sentence follows another, which helps in understanding the logical flow between sentences .
- For spam detection, this ability is useful when considering the relationship between different parts of an email, such as the subject line and the body.

1.5.5.2 Fine-tuning Phase

After the pre-training phase, BERT is fine-tuned on a specific task, such as spam email detection. During fine-tuning, BERT's general language understanding is adapted to the specific requirements of the task through supervised learning on a labeled dataset. The fine-tuning phase involves the following steps:

1. Task-Specific Input Processing**:**

- For spam email detection, the input text (which could include the subject line and body of an email) is processed through the BERT model.
- The same token, segment, and position embeddings used during pre-training are applied to convert the text into a format that BERT can process.

2. Classification Layer Addition:

- A task-specific classification layer is added on top of the BERT model. This typically involves adding a fully connected (dense) layer that outputs logits corresponding to the classes (e.g., spam or not spam) .
- For binary classification tasks like spam detection, the output layer often includes a sigmoid function to produce a probability score for each class.

3. Training and Fine-Tuning:

- During fine-tuning, the entire BERT model, along with the newly added classification layer, is trained on the labeled spam detection dataset.
- The model's parameters are updated to minimize the loss function (e.g., binary cross-entropy loss), which measures the difference between the predicted probabilities and the actual labels (spam or not spam) .
- Fine-tuning allows BERT to learn the specific language patterns, keywords, and contextual cues that are indicative of spam emails.

4. Prediction and Inference:

- Once fine-tuning is complete, the model can be used to predict whether an incoming email is spam or not. The input text is passed through the BERT model, which processes it to generate a context-aware representation.
- The classification layer then uses this representation to output a probability score for each class, indicating the likelihood that the email is spam, (Devlin et al., 2019).

Gmail and outlook likely employs a combination of various machine learning techniques, including neural networks and deep learning models like BERT, to achieve high accuracy in spam detection. The choice of algorithms may vary over time as new techniques are developed and existing models are refined to improve performance

1.6 Literature Review

Spam email detection has been a prominent research area in natural language processing (NLP) and machine learning (ML) due to the growing challenges posed by spam in email communication. This section reviews significant contributions in the field, including traditional machine learning methods and advanced deep learning approaches, and highlights their relevance to the present study.

The Naive Bayes classifier is widely recognized for its simplicity and effectiveness in spam detection. For instance, (Androutsopoulos, 2000), applied the Naive Bayes algorithm to a large dataset of spam emails and found it to be particularly effective, achieving a recall of 96% and precision of 94%. This work highlighted the robustness of the Naive Bayes classifier in text classification tasks, despite its assumption of feature independence.

Support Vector Machines (SVMs) have also maintained their popularity in spam detection due to their ability to handle high-dimensional data. (Guo, 2020), introduced a hybrid SVM model that incorporated feature selection and ensemble techniques to enhance spam detection accuracy. The optimized SVM model achieved an impressive accuracy of 98%, with precision and recall rates of 97% and 96%, respectively, reinforcing SVM's position as a leading choice for spam classification.

Meanwhile, logistic regression remains a viable option for spam detection, as demonstrated by (Li, 2019), who explored its effectiveness when combined with various feature extraction techniques. The study showed that with optimized features, logistic regression achieved an accuracy of 93%, with precision and recall metrics of 91% and 90%, respectively, underscoring its adaptability and potential for achieving high accuracy rates.

Decision trees, particularly when used in ensemble methods, have seen renewed interest in recent years. (Zhang, 2022), proposed a decision tree-based method that incorporated semantic information from word embeddings, achieving an accuracy of 94%, with precision and recall rates of 92% and 91%, respectively, on a benchmark spam dataset. This approach demonstrated the potential of decision trees to capture complex patterns in spam emails.

Our study differs from previous work in several key aspects. First, we use a more recent and diverse dataset, ensuring that our results are applicable to the current landscape of email spam. Moreover, we perform a comparative analysis of various traditional algorithms, including Naive Bayes, Logistic Regression, Decision Trees, and SVM, against the state-of-the-art BERT model. This comparison allows us to evaluate how these traditional methods measure up against BERT, particularly in terms of accuracy, precision, and recall, providing a comprehensive understanding of their performance in contemporary spam detection scenarios.

Chapter 2

2.1 Dataset

The “Email Spam Detection” dataset, created by Zeeshan Younas[1], is a comprehensive collection of emails classified as either spam or non-spam. Its primary purpose is to aid in the development and evaluation of machine learning models aimed at detecting spam emails. Hosted on Kaggle, a reputable and widely recognized platform in the data science and machine learning community, this dataset was uploaded in May 2024 and downloaded for this project in June 2024, ensuring that the data is both current and relevant to the study's objectives. Kaggle is known for providing high-quality datasets contributed by users worldwide, and the recent upload of this dataset makes it particularly suited for tackling current issues in email filtering. Furthermore, the dataset mirrors real-world email data, offering a practical foundation for developing models that can be directly implemented in actual email filtering systems. This real-world applicability ensures that the insights and models derived from this dataset will be effective in practical scenarios.

2.1.1 Dataset Details

Ham Messages: These messages are likely normal communications that do not contain any malicious content or advertising.
Spam Messages: These messages typically contain promotional content, phishing attempts, or other forms of unsolicited communication.

Entries and Columns: The dataset contains 5,572 entries with two columns:

- Category**:** Indicates whether the email is 'ham' (not spam) or 'spam'.
- Messages: Contains the text of the email

[1] https://www.kaggle.com/datasets/zeeshanyounas001/email-spam-detection/discussion?sort=hotness

- Unique Messages : The Massage's column contains 5,169 unique messages out of 5,572 entries. This indicates the presence of duplicate messages.
- Presence of Duplicates : There are 403 duplicate messages in the dataset (5,572 total messages - 5,169 unique messages = 403 duplicates). Duplicate messages can skew the analysis and model training. It's essential to decide whether to keep these duplicates based on their relevance to the task at hand. In some cases, duplicates may be useful, while in others, they may introduce bias.

  As shown in **Figure 2** ham appears 4,825 times, accounting for approximately 86.6% of the dataset, spam appears 747 times, accounting for approximately 13.4% of the dataset.

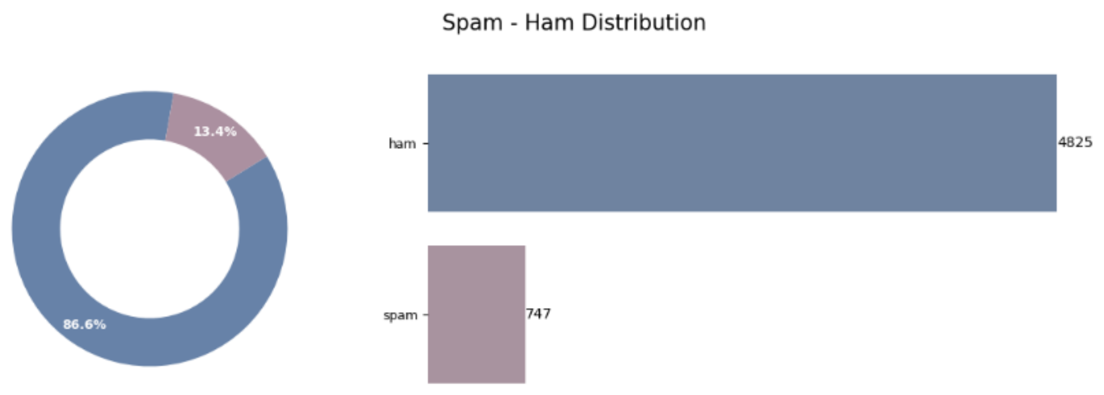


Figure 2 : Spam - ham distribution

2.1.2 Dataset Imbalance

The dataset exhibits an imbalance, with a significantly higher number of 'ham' messages (86.6%) compared to 'spam' messages (13.4%). This imbalance can affect the performance of machine learning models, which might become biased towards the majority class. Handling such imbalances is crucial for developing robust and accurate spam detection systems.

2.2 Tools and Libraries Used

2.2.1 Programming Language: Python 3.8

Python 3.8 was selected as the primary programming language for this project due to its extensive support for data handling, statistical analysis, and machine learning. Python's rich ecosystem of libraries and frameworks made it particularly well-suited for developing and evaluating machine learning models, especially in the domain of natural language processing (NLP), (Millman, 2011).

2.2.2 Libraries and Frameworks

- Pandas: Pandas was utilized for data manipulation and exploratory data analysis (EDA). Its robust data structures, particularly DataFrames, allowed for efficient handling of large datasets, facilitating tasks such as data cleaning, transformation, and aggregation, (McKinney, 2010).
- NumPy: NumPy was essential for scientific computing tasks, providing support for large, multi-dimensional arrays and matrices. Its array operations were crucial for performing mathematical computations, enabling efficient numerical processing that underpinned the machine learning algorithms, (Van der Walt, 2011).
- Scikit-learn: Scikit-learn was the core library for machine learning tasks, offering a comprehensive suite of tools for data preprocessing, model building, and evaluation. It provided implementations for various classification algorithms, including Logistic Regression, Support Vector Machines (SVM), and Naive Bayes, which were pivotal in this project's spam detection efforts, (Pedregosa, 2011).
- NLTK (Natural Language Toolkit): NLTK was extensively used for natural language processing, offering a range of tools for text processing tasks such as tokenization, stemming, and stop word removal. These operations were critical for preparing textual data for machine learning models, ensuring that the data was in a suitable format for analysis, (al, 2009).

- Matplotlib and Seaborn: Matplotlib and Seaborn were instrumental in creating visualizations that aided in data exploration and model performance analysis. These libraries provided tools to generate a wide variety of plots, such as histograms, scatter plots, and bar charts, which helped in understanding data distributions and relationships, as well as in interpreting the results of the models, (Hunter, 2007).
- Regex (re module): The Python re module was integrated for pattern matching and text cleaning operations. Regular expressions were used to identify and clean unwanted patterns in the text, such as special characters, ensuring uniformity and consistency in the textual data preprocessing pipeline, (Foundation, 2024).
- PyTorch and Transformers: PyTorch, in conjunction with the Transformers library, was employed for training deep learning models like BERT. These libraries facilitated the implementation and fine-tuning of BERT, allowing for the effective handling of text sequences and leveraging pre-trained models for spam detection tasks, (Paszke et al, 2019).
- Imbalanced-learn: Imbalanced-learn was crucial for handling imbalanced datasets using techniques such as SMOTE (Synthetic Minority Over-sampling Technique). This library ensured that the models were trained on balanced datasets, mitigating bias and improving classification performance on minority classes, (Lemaître et al, 2017).
- sklearn.utils: The resample function from the sklearn.utils module was instrumental in handling class imbalance by providing simple yet effective resampling techniques. It allowed for oversampling the minority class or undersampling the majority class, which was crucial for creating balanced training datasets.
- importlib: The importlib module was used for dynamically importing modules during runtime, allowing for flexible and efficient management of dependencies. This feature was particularly useful for loading modules or packages only when needed, reducing memory usage and improving overall performance in large projects.
- itertools: The itertools module played a key role in working with iterators, providing powerful tools for efficiently iterating over data. Its functions like product() were used for Cartesian products and generating combinations, which streamlined processes such as hyperparameter tuning and grid search in machine learning tasks

- joblib: The joblib library was essential for serializing and deserializing models, allowing for efficient saving and loading of machine learning models. This functionality ensured that trained models could be stored and reused without the need for retraining, saving both time and computational resources[2].
- ABC (Abstract Base Classes): The ABC module was used to create abstract classes like AbstractModel, providing a template for developing various machine learning models. This approach promoted code reusability and consistency across different model implementations, (Python Software Foundation., 2024).
- OS module: The OS module was used for environment settings, particularly in managing file paths and system-level operations during data loading and model training processes, (Python Software Foundation., 2024)
- TensorFlow: TensorFlow was employed for training models that relied on this library. While PyTorch was the primary deep learning framework, TensorFlow was also utilized for certain models, providing flexibility and compatibility with specific architectures, (Abadi et al, 2016).
- Plotly Graph Objects: plotly.graph_objects was used for creating interactive, high-quality visualizations. This module allowed the customization of plots and figures, offering flexibility in constructing graphs such as line charts, bar plots, and scatter plots. The integration of go.Figure and trace-based design provided an intuitive approach to displaying results interactively within Jupyter notebooks or other web-based environments, (Plotly Technologies INC, 2015).
- WordCloud: This library is used for generating word clouds from text data. It visually represents the most frequent words in a dataset, where the size of each word corresponds to its frequency. WordCloud is especially useful for quickly identifying important terms in spam and ham messages before and after preprocessing.
- PCA (Principal Component Analysis): PCA is a technique for reducing the dimensionality of datasets, increasing interpretability while minimizing information loss. In this project, it

[2] https://scikit-learn.org/

was applied to visualize the distribution of spam and ham messages before and after applying the SMOTE technique, helping in the analysis of class balance.

- Streamlit: Streamlit was used for building the web application interface for the spam detection project. It provided a simple yet powerful framework to create interactive web apps directly from Python scripts. The import streamlit as st command enabled the integration of various UI components such as input fields, buttons, and output displays, facilitating real-time interaction with the trained machine learning models.

2.2.3 Data Processing Tools

- Microsoft Excel: Microsoft Excel was employed during the initial stages of data inspection and preliminary preprocessing. Excel provided a user-friendly interface for a preliminary overview of the dataset, allowing for basic data cleaning tasks, such as handling missing values and simple transformations, before more advanced processing in Python.

2.2.4 Visualization Tools

- Matplotlib and Seaborn: These libraries were crucial in visualizing data distributions and transformations throughout the preprocessing phase. Matplotlib and Seaborn enabled the creation of detailed plots that illustrated the distribution of data before and after applying SMOTE (Synthetic Minority Over-sampling Technique) and displayed the space count in text data both before and after the removal of extra spaces. These visualizations were essential for understanding the impact of data preprocessing steps and ensuring data consistency before model training.

2.2.5 Machine Learning Libraries

- Scikit-learn: Scikit-learn served as the backbone for implementing the machine learning models in this study. It provided a range of algorithms, including Logistic Regression, Support Vector Machines (SVM), and Naive Bayes, each of which was evaluated for its effectiveness in classifying emails as spam or non-spam. Scikit-learn's easy-to-use API facilitated rapid experimentation and tuning of these models.

2.2.6 Text Processing Tools

- NLTK (Natural Language Toolkit): NLTK, coupled with custom text preprocessing functions, was crucial in preparing the textual data for analysis. NLTK's comprehensive tools for text tokenization, stop word removal, and stemming ensured that the text data was effectively processed, reducing dimensionality and focusing on the most informative features for model training, (Bird, 2006).

Stopwords in Text Processing: Stopwords are common words (e.g., "and", "the", "is") that typically do not add significant meaning to the content and can be removed during text preprocessing. NLTK provides a convenient list of stopwords across various languages, facilitating the preparation of text data by filtering out these irrelevant words. This step was essential in focusing the model on more meaningful words, thereby improving the efficiency and effectiveness of the spam detection system, (Loper, et al, 2002).

2.4 Oversampling

Oversampling is a technique used to handle imbalanced datasets by increasing the number of instances in the minority class to match the majority class. Unlike SMOTE, which generates synthetic samples by interpolating between data points, oversampling simply replicates existing instances in the minority class. While this method can increase the risk of overfitting, it is particularly suitable for scenarios where generating synthetic data may not be appropriate, such as with textual data.

In this study, oversampling was used specifically for the BERT model due to the limitations of SMOTE in dealing with text-based inputs. SMOTE generates numeric data points, which are incompatible with BERT's requirement for textual input. Since BERT processes text as embeddings, the integrity of the textual data must be maintained throughout the model training process. Therefore, oversampling was chosen to balance the dataset without altering the text data or generating synthetic numeric values. By oversampling the minority class (spam emails), the model was trained on a balanced dataset, leading to improved classification performance in detecting both spam and non-spam emails, (He, 2009).

2.3 Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE (Synthetic Minority Over-sampling Technique) is a powerful method used to address the challenge of imbalanced datasets in machine learning, where one class significantly outnumbers another, leading to biased models that often perform poorly on the minority class. Unlike simple oversampling, which duplicates existing minority class samples and can result in overfitting, SMOTE generates new, unique samples by interpolating between the minority class samples and their nearest neighbors, thereby reducing the risk of overfitting. This process involves selecting a random sample from the minority class, identifying its k-nearest neighbors, and creating synthetic samples by interpolating between the selected sample and a randomly chosen neighbor. These synthetic samples are then added to the dataset, increasing the number of minority class instances and effectively balancing the dataset.

SMOTE is also superior to undersampling, which reduces the number of majority class samples and can lead to the loss of valuable information. By using SMOTE in this study, a more balanced and effective model was developed, leading to improved performance metrics such as recall and F1-score, making it a preferred choice over other methods like simple oversampling or undersampling, (Chawla, 2002) .

In this study, as shown in Figure 4, SMOTE was employed to balance the dataset by generating synthetic samples for the minority class (spam emails). The 3D plot illustrates the distribution of data points after applying SMOTE, where the red points represent the new spam samples. Notably, there is some overlap with the original data points, visualized by the green points, and a broader spread of the synthetic samples in space. This scatter is a characteristic behavior of SMOTE, which creates new data between existing minority class points. By balancing the dataset, this step prevented the machine learning models from being biased towards the majority class (ham emails), allowing them to improve their performance in detecting spam emails. The balanced dataset played a crucial role in enhancing the robustness and accuracy of the spam detection models, ultimately leading to more reliable results.

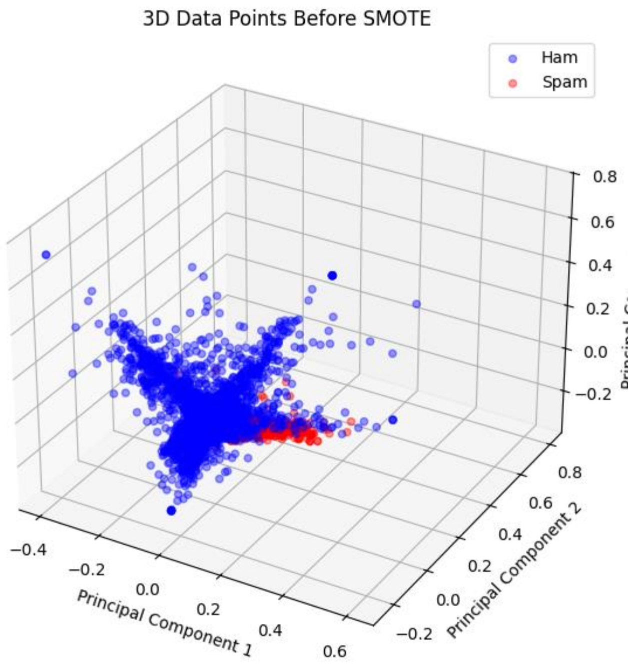


Figure 3 : Imbalanced dataset before applying smote

the axes labeled "Principal Component 1," "Principal Component 2," and "Principal Component 3" represent the top three dimensions derived from Principal Component Analysis (PCA). These components capture the most significant patterns in the dataset's variance. Principal Component 1 accounts for the highest variance in the data, followed by Principal Component 2 and Principal Component 3, which capture additional, yet progressively smaller, variances. By reducing the dataset to these three dimensions, the 3D plot provides a clear visualization of the spread and overlap between the original and SMOTE-generated samples, aiding in the interpretation of the balanced dataset.

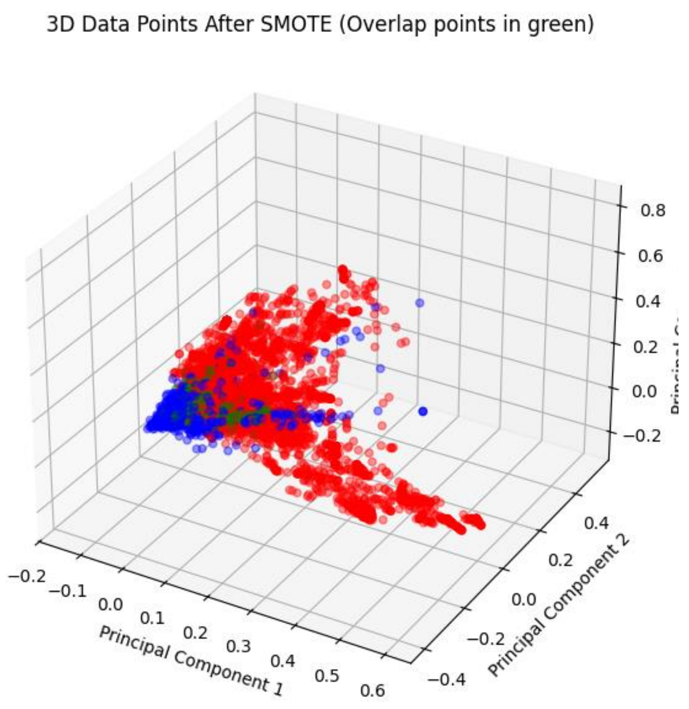


Figure 4 : balanced dataset after applying smote

2.5 WordCloud

In this study, WordCloud visualizations were utilized to provide an initial understanding of the textual data, particularly highlighting the frequency and prominence of words in the dataset both before and after preprocessing, (Heimerl et al, 2014). The WordCloud for spam messages before preprocessing as shown in **Figure 5,** revealed a cluttered and noisy landscape dominated by words such as "FREE," "CALL," "WIN," and "MOBILE," which are typical indicators of spam content. However, this raw visualization also included numerous irrelevant terms, special characters, and inconsistent word forms, underscoring the need for thorough preprocessing.

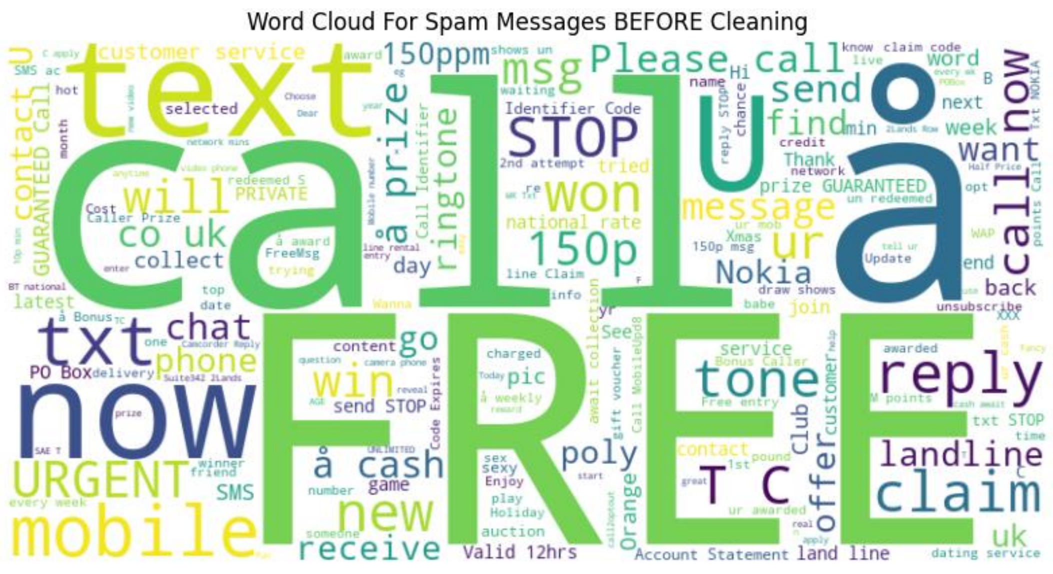


Figure 5 : Word Cloud for Spam Messages BEFORE Cleaning

After applying text preprocessing techniques—including tokenization, stop word removal, and stemming—the WordCloud for spam messages as shown in **Figure 6**, became significantly cleaner and more focused, with noise substantially reduced and the most prominent spam-indicating terms more clearly emphasized. Similarly, WordClouds generated for ham messages before and after preprocessing demonstrated a shift from a cluttered mix of everyday terms to a more refined and balanced distribution of meaningful words that better represent non-spam communication. These WordClouds not only provided valuable insights into the dataset but also illustrated the critical importance of preprocessing in enhancing data quality. By removing extraneous noise and standardizing the text, the preprocessing ensured that the machine learning models could focus on the most informative features, ultimately improving their ability to accurately distinguish between spam and ham emails.

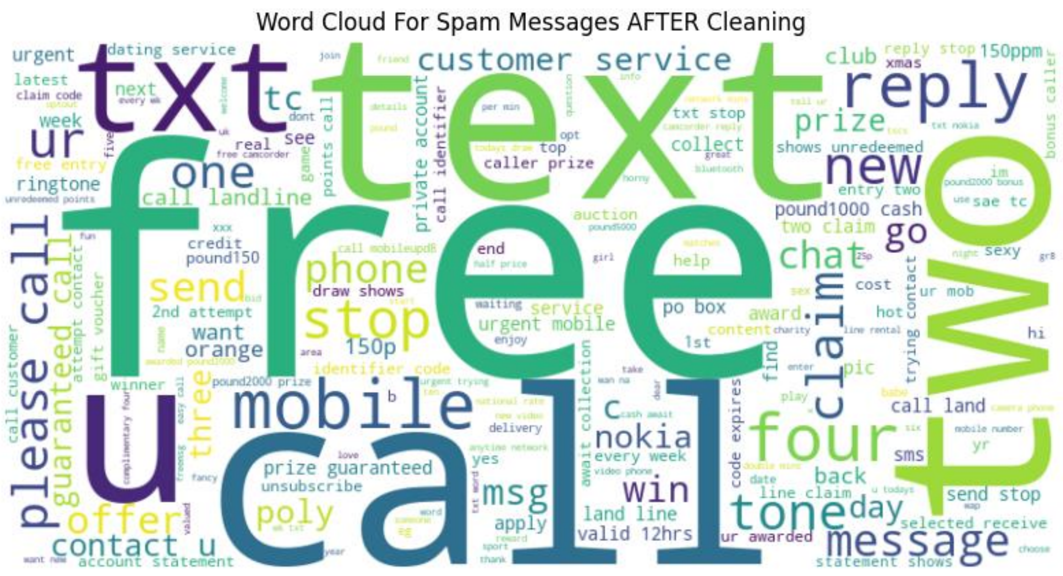


Figure 6 : Word Cloud For Spam Messages After Cleaning

Figure 7 displays the WordCloud for ham messages before preprocessing, where common words like "it," "will," "got," and "know" dominate the visualization. These terms, while frequent, do not provide significant insight into the unique characteristics of non-spam messages, as they include many irrelevant and frequently occurring words. This raw data also contains extraneous words, special characters, and various inconsistent forms of the same word, demonstrating the need for comprehensive text preprocessing.

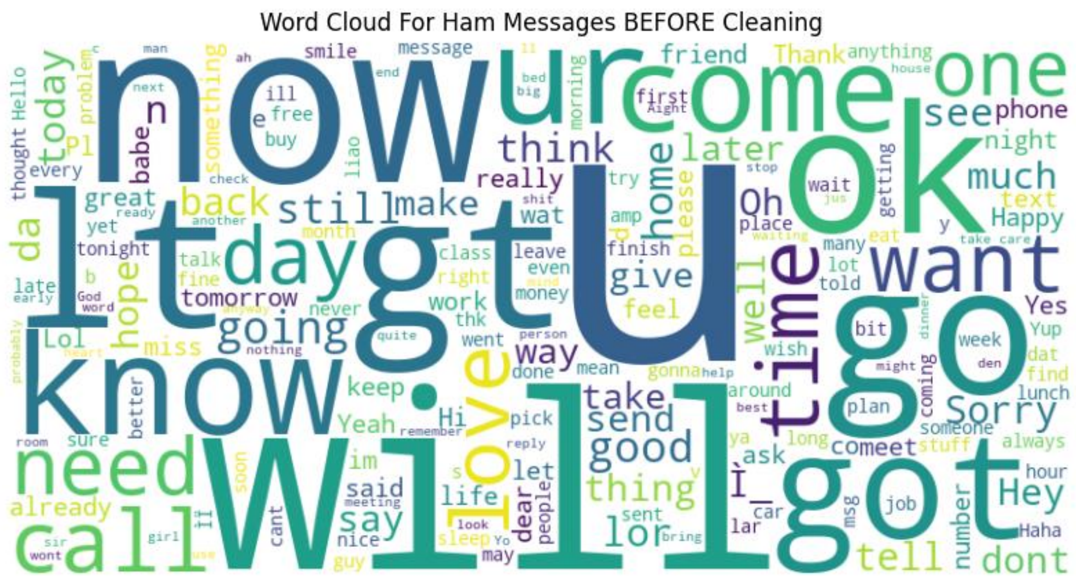


Figure 7 : Word Cloud For Ham Messages Before Cleaning

After applying text preprocessing techniques, including tokenization, removal of stop words, and stemming, **Figure 8** shows a markedly different WordCloud for ham messages. This cleaner and more structured representation focuses on words with higher significance for ham classification, such as "time," "need," "love," and "later." The preprocessing effectively filtered out irrelevant noise, allowing the more meaningful terms to surface, which are critical for improving the accuracy of machine learning models. These refined WordClouds emphasize the importance of preprocessing in creating higher-quality data, ultimately aiding in the accurate distinction between ham and spam messages. By transforming the raw, cluttered text into a well-organized form, the models can more effectively identify the relevant patterns necessary for classification.

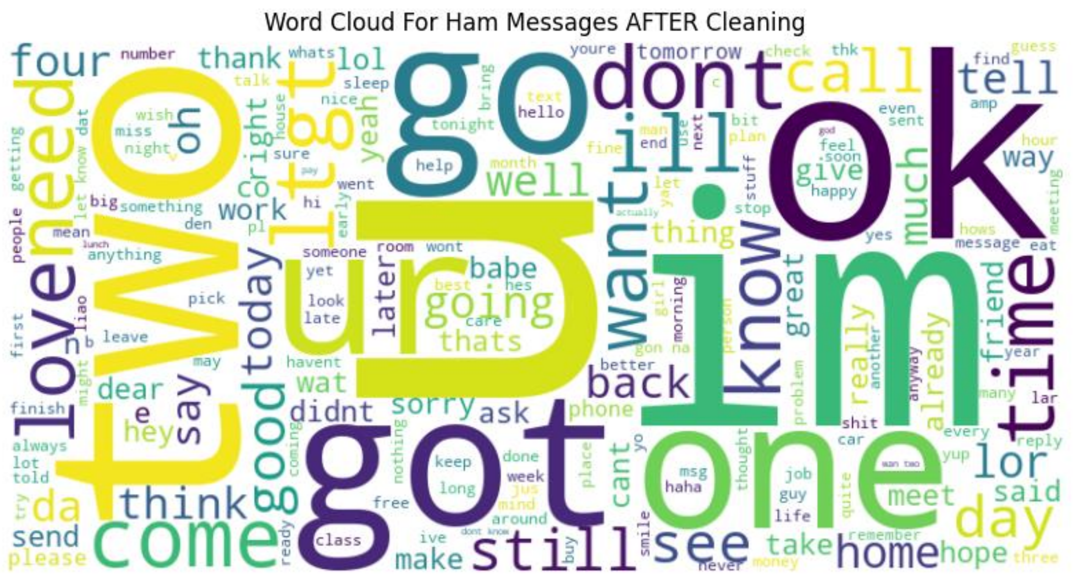


Figure 8 : Word Cloud For Ham Messages After Cleaning

2.6 Feature selection algorithms

Feature selection is a crucial step in preparing data for machine learning models, particularly when dealing with high-dimensional datasets like text data. It involves selecting the most informative features (in this case, words or tokens) to improve model performance by reducing noise, computational cost, and overfitting. Below are key techniques for feature selection in Natural Language Processing (NLP):

2.6.1 Bag-of-Words

is a technique used in Natural Language Processing (NLP) to represent a text document as a collection of words, disregarding the order or grammar of those words. It's like creating a bag of unique words that appear in the text, similar to how a bag might hold various items without any specific organization. How BoW Works:

1. Text Preprocessing: Convert the text into a consistent format by performing lowercasing, removing punctuation, stop words, and other irrelevant characters.
2. Vocabulary Creation: Compile a list of all unique words present in the corpus (the collection of all documents).

3. Vectorization: Represent each document as a vector of word counts. Each element in the vector corresponds to the frequency of a specific word in the document.

   Consider the following words extracted from emails after preprocessing, the table shows the Bag of words of each Word

Word	**BoW**
mobile	112
free	178
update	17
camera	22
cash	61
cried	1
anything	69

Table 1 : BoW of Selected Words

2.6.2 TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF, or Term Frequency-Inverse Document Frequency, is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus). It is a widely used technique in information retrieval and text mining for transforming text into numerical features that can be used by machine learning algorithms.

Components of TF-IDF:

1. Term Frequency (TF):
   Term Frequency (TF) measures how frequently a term appears in a document. The assumption is that the more a term appears in a document, the more significant it is within that document.

TF is calculated as:

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

2. Inverse Document Frequency (IDF):

   Inverse Document Frequency (IDF) measures how important a term is in the corpus. The assumption is that if a term appears in many documents, it is less informative, and its significance should be reduced.

   - IDF is calculated as:

$$\text{IDF}(t,D) = \log\left(\frac{\text{Total number of documents } N}{\text{Number of documents containing term } t}\right)$$

3. TF-IDF Calculation:

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

Normalization in TF-IDF:

Normalization in TF-IDF typically involves adjusting the magnitude of the TF-IDF vectors so that they conform to a specific norm, commonly the L2 norm. The L2 norm (also known as Euclidean norm) ensures that the sum of the squares of the vector elements equals 1, which standardizes the length of vectors without altering their direction.

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

This represents the square root of the sum of the squared TF-IDF scores for all terms in the document.

Normalize Each Element: Each element vi of the vector v is divided by the L2 norm:

$$v_i' = \frac{v_i}{\|\mathbf{v}\|_2}$$

where v′i is the normalized TF-IDF score for term i.

- normalization helps maintain the integrity of the data and improves the reliability of analyses and comparisons performed on TF-IDF vectors, (Ramos, 2003).

Example of TF-IDF Calculation:

Consider the following words extracted from emails after preprocessing, The table shows the Term Frequency (TF), Inverse Document Frequency (IDF), and the combined TF-IDF score for each word.

WORD	TF	IDF	TF-IDF
mobile	0.125	4.894955	0.334
free	0.125	4.281851	0.292166
update	0.125	6.831296	0.466123
camera	0.0625	6.448304	0.219995
cash	0.047619	5.428472	0.168064
cried	0.066667	8.846199	0.408367
anything	0.5	5.319838	0.748085

Table 2 : Term Frequency-Inverse Document Frequency (TF-IDF) Values of Selected Words

Use in Spam Detection:

In the context of spam email detection, TF-IDF can be used to convert emails into numerical feature vectors. These vectors can then be used as input for traditional machine learning models, such as Naive Bayes, SVM, or Logistic Regression, to classify emails as spam or ham. TF-IDF helps in identifying significant terms that are common in spam emails but not in legitimate ones, thereby aiding the classification process, (Sahami e. a., 1998).

BERT’s Embedding Capabilities:

BERT, a transformer-based model pretrained on vast amounts of text data, excels in capturing intricate semantic meanings and contextual relationships within language. Unlike TF-IDF, which computes the importance of words based on their occurrence frequency across documents, BERT learns contextual embeddings that encode the meaning of words and their relationships in sentences bidirectionally. This approach allows BERT to understand nuances such as word sense disambiguation and syntactic structures, which are crucial for tasks requiring deeper linguistic understanding, (Devlin et al., 2019).

Advantages Over TF-IDF:

When fine-tuning BERT for specific NLP tasks like sentiment analysis, text classification, or named entity recognition, the need for manually extracting TF-IDF features diminishes. BERT embeddings alone often outperform TF-IDF in terms of accuracy and performance metrics. This superiority stems from BERT’s ability to leverage contextual information across sentences and documents, providing richer and more nuanced representations of textual data compared to the static and potentially less discriminative features derived from TF-IDF, (Devlin et al., 2019).

2.7 Preprocessing and Feature Engineering

2.7.1 Explanation and Functionality

1. Initialization:

The “TextCleaner” class is initialized with the raw email text. During initialization, the text is converted to lowercase to ensure uniformity. This step helps the model treat words like "Spam" and "spam" as the same word, reducing the number of unique words the model needs to learn.

2. Replacing URLs:

The “replace_urls” method replaces URLs with the placeholder 'URL'. URLs often do not contribute meaningful information for spam detection and can introduce noise into the dataset. By standardizing them, we reduce variability and focus on the textual content.

3. Replacing Money Symbols:

The “replace_money_symbols” method replaces common currency symbols with their respective text equivalents. This step helps in recognizing patterns related to financial scams and phishing attempts, which are common in spam emails.

4. Replacing Digits with Text:

The “replace_digits_with_text_numbers” method converts numerical digits to their word equivalents. This normalization helps the model better understand numbers in the context of the text, enhancing its ability to detect numerical patterns associated with spam.

5. Removing Non-Alphanumeric Characters:

The “remove_non_alphanumeric” method removes all characters except for letters and numbers. This helps in focusing on the core textual content and removing extraneous symbols that do not contribute to the analysis.

6. Removing Extra Spaces

The “remove_extra_spaces” method eliminates multiple consecutive spaces, standardizing the text format and ensuring consistency in word spacing.

Visualizing Space Counts: Before applying the "remove_extra_spaces" method, it is informative to visualize the distribution of space counts across the dataset. This visualization provides insights into the variability and prevalence of excessive spaces in the raw email texts.

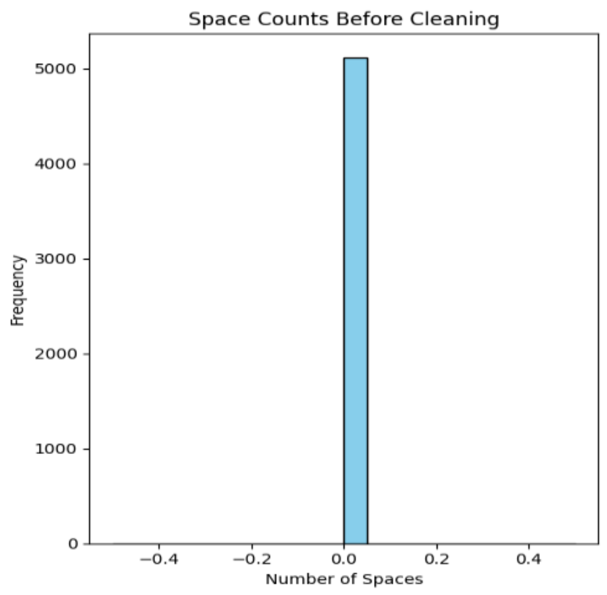


Figure 9 : Distribution of Space Counts Before preprocessing

Figure 9 illustrates the frequency distribution of space counts across a sample of email texts before any preprocessing steps are applied. The histogram reveals a range of space counts, highlighting instances where multiple consecutive spaces may exist within the text.

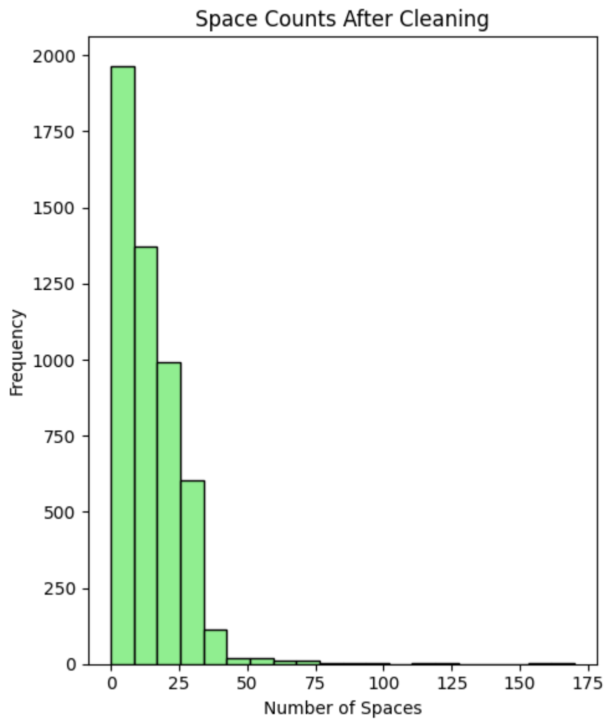


Figure 10 : Distribution of Space Counts After preprocessing

Figure 10 depicts the distribution of space counts after applying the "remove_extra_spaces" method. Noticeably, the histogram shows a reduction in the frequency of higher space counts, indicating successful standardization of word spacing.

Comparing these visualizations underscores the impact of space management on text normalization. By removing extra spaces, we mitigate inconsistencies in text formatting, thereby improving the quality and uniformity of the dataset for subsequent analysis.

7. Removing Stop Words:

The “remove_stop_words” method filters out common words (like "and", "the", "is") that do not add significant meaning to the content. This step reduces the dataset size and enhances the model’s focus on more informative words.

8. Combining All Steps:

The “clean” method sequentially applies all the cleaning steps to the text, returning a fully normalized and cleaned version of the email content.

2.7.2 Application to Dataset

- **Loading the Dataset:** The dataset is loaded from a CSV file into a Pandas DataFrame.
- **Cleaning the Emails:** Each email is cleaned using the TextCleaner class, with the cleaned text replacing the original in the DataFrame.
- **Handling Missing Values:** Rows with missing email content are removed to ensure data quality.
- **Label Encoding:** The 'Category' column labels are converted to numerical representations.
- **Handling Duplicates:** Duplicate messages are removed to prevent skewing the analysis.
- **Word Frequency Analysis:** This step identifies vocabulary differences between spam and legitimate emails.
- **Text Vectorization:** The ‘TfidfVectorizer’ from Scikit-learn is used to transform text into numerical features using TF-IDF, which captures the importance of words in the context of the entire dataset.

- **Balancing the Dataset:** The SMOTE (Synthetic Minority Over-sampling Technique) algorithm is applied to balance the dataset by generating synthetic samples for the minority class.
- **Word Frequency Analysis:** Word frequency analysis identifies the most common words in spam and ham messages, with bag-of-words representations created for all messages, spam, and ham separately.
- **Word Cloud Generation:** Word clouds visualize the most frequent words in the entire dataset and specifically in spam and ham messages, aiding in a quick visual understanding of the prominent terms.
- **Saving the Cleaned Dataset:** The cleaned dataset is saved to a new CSV file.

The preprocessing and normalization steps implemented in the 'TextCleaner' class are crucial for preparing email data for machine learning analysis. By standardizing text format and removing irrelevant content, we enhance the model's ability to accurately detect spam emails. This rigorous preprocessing pipeline ensures that the model can focus on meaningful patterns, leading to improved performance and reliability in spam detection.

Chapter 3

3.1 Evaluation metrics

Evaluation metrics are essential for assessing the performance of machine learning models, particularly in classification tasks like spam email detection. The choice of metrics depends on the specific goals and characteristics of the problem. For spam detection, we focus on several key metrics that provide a comprehensive view of the model's performance. Below are the main evaluation metrics used in this research:

3.1.1 Accuracy

Accuracy is the ratio of correctly predicted instances to the total instances. It is a widely used metric for classification problems. However, in the context of spam detection, accuracy alone can be misleading due to the class imbalance (i.e., more ham emails than spam emails).

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

3.1.2 Precision

Precision measures the proportion of correctly predicted positive instances (spam emails) out of all instances predicted as positive. High precision indicates that the model is good at minimizing false positives.

$$\text{Precision} = \frac{TP}{TP+FP}$$

3.1.3 Recall

Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances out of all actual positive instances. High recall indicates that the model is good at identifying spam emails and minimizing false negatives.

$$\text{Recall} = \frac{TP}{TP+FN}$$

3.1.4 F1 Score

F1 Score is the harmonic mean of precision and recall. It provides a single metric that balances the trade-off between precision and recall, making it particularly useful when dealing with imbalanced datasets.

$$\text{F1 Score} = 2 \times \frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$$

3.1.5 Confusion Matrix

A Confusion Matrix is a table used to evaluate the performance of a classification model. It provides a detailed breakdown of the model's predictions, showing the counts of true positives, true negatives, false positives, and false negatives.

	Predicted Positive (Spam)	**Predicted Negative (Ham)**
Actual Positive (Spam)	True Positive (TP)	False Negative (FN)
Actual Negative (Ham)	False Positive (FP)	True Negative (TN)

Table 3 : Confusion Matrix

Where:

- TP (True Positives): Number of spam emails correctly identified as spam.
- TN (True Negatives): Number of ham emails correctly identified as ham.

- FP (False Positives): Number of ham emails incorrectly identified as spam.
- FN (False Negatives): Number of spam emails incorrectly identified as ham.

3.1.6 Evaluation Metric Selection for Spam Detection

In the context of spam email detection, precision and recall are often given more emphasis due to the following reasons:

- Precision: is important to minimize the number of legitimate emails (ham) incorrectly classified as spam (false positives). High precision ensures that users do not miss important emails.
- Recall: is crucial to identify as many spam emails as possible (true positives), reducing the chances of spam emails slipping through the filter (false negatives).

The F1 Score, which balances precision and recall, provides a useful single metric to evaluate the model's overall effectiveness, while the confusion matrix offers a comprehensive view of the classification results.

By employing these evaluation metrics, we can thoroughly assess the performance of the BERT-based spam detection model and compare it with traditional machine learning approaches. This comprehensive evaluation helps ensure that the chosen model effectively balances the trade-offs between different aspects of spam email detection, (Sokolova, 2009).

3.2 Hyperparameters in Machine Learning

In the context of spam email detection, hyperparameters play a critical role in optimizing a model's ability to accurately classify emails as spam or ham. Proper tuning of hyperparameters can significantly improve precision, ensuring that legitimate emails are not incorrectly classified as spam, while also enhancing recall by correctly identifying most spam emails. Additionally, hyperparameter tuning helps balance the model, achieving an optimal trade-off between precision and recall, which is particularly important in handling imbalanced datasets such as those encountered in spam detection. (Sculley. et al., 2007) (Brownlee, 2018)

Hyperparameters are external configurations set before the learning process begins, which govern the behavior and performance of machine learning models. Unlike model parameters, which are learned during training, hyperparameters are predefined and influence the model's ability to learn from data. Choosing the right set of hyperparameters can significantly impact a model's accuracy, precision, recall, and overall performance, (Goodfellow, 2016).

Common hyperparameters in machine learning include:

- **Learning Rate:** Controls how much the model adjusts its parameters with each iteration. A lower learning rate can lead to better accuracy but slower training.
- **Regularization Parameters (e.g., L2):** also known as Ridge regression. L2 regularization works by adding a penalty term to the loss function, proportional to the square of the magnitude of the model's coefficients. This penalty discourages the model from assigning excessively high values to any single coefficient, promoting a more balanced weight distribution across features.
- **Number of Epochs:** Specifies the number of times the learning algorithm will work through the entire training dataset.
- **Batch Size:** Determines the number of training examples utilized in one iteration.
- **Kernel Type (for SVM):** Defines the type of kernel function to be used in the algorithm (e.g., linear, polynomial, radial basis function).
- **Max Depth (for Decision Trees):** Limits the maximum depth of the tree, helping to control overfitting.
- **Dropout Rate (For BERT):** A regularization technique used to prevent overfitting in neural networks by randomly setting a fraction of input units to zero during training.

3.2.1 Hyperparameter Tuning

Hyperparameter tuning is the process of finding the optimal set of hyperparameters that yield the best performance for a specific model. This process typically involves three main methods: Grid Search, where different combinations of hyperparameters are systematically tested to find the best combination; Random Search, which selects combinations randomly rather than exhaustively testing all possibilities; and Bayesian Optimization, a more sophisticated method that builds a probabilistic model to identify the best set of hyperparameters more efficiently (Géron). In this research, Random Search was chosen for hyperparameter tuning due to its advantage of being more computationally efficient compared to Grid Search, while still providing a good exploration of the hyperparameter space. This strategy was applied to each of the machine learning models to optimize their performance in spam detection, (Brownlee, 2018).

In our study, hyperparameter tuning was performed for each of the machine learning models to optimize their performance in spam detection. The following tuning strategies were used:

- **BERT:** Fine-tuning involved adjusting the learning rate, number of epochs, and dropout rate to achieve the best balance between accuracy and recall.
- **Support Vector Classifier (SVC):** The kernel type, regularization parameter (C), and gamma were fine-tuned to maximize the model's precision and recall.
- **Logistic Regression:** The regularization parameter (C) was adjusted to improve the model's ability to generalize from the training data.
- **Decision Tree:** The max depth and minimum samples split were fine-tuned to balance model complexity and performance.
- **Naive Bayes:** Although Naive Bayes has fewer hyperparameters, the smoothing parameter (alpha) was tuned to enhance its performance.

3.3 K-Fold Cross-Validation

K-fold cross-validation is a robust technique used to evaluate the performance of a model by dividing the dataset into k equal-sized folds. Its advantages include reduced bias, as training and validating the model on different subsets provides a more comprehensive evaluation of its performance and better generalization, which helps in understanding how well the model generalizes to unseen data by testing the model on multiple validation sets, (Arlot, 2010).

Process :

- The dataset is split into *k* subsets (or folds).
- The model is trained *k* times, using *k-1* folds for training and the remaining fold for validation in each iteration.
- Each fold serves as the validation set exactly once, and the model's performance is evaluated across all folds, (Kohavi R. , 1995).

Impact on Results:

- Using 5-fold cross-validation allows to assess the performance metrics of the models (Accuracy, Precision, Recall, F1-Score) more reliably.
- It helps in identifying the best performing model configuration by averaging results over 5 folds, thus giving a more stable estimate of model performance (Kohavi R. , 1995)

Chapter 4

4.1 Introduction to Results

This chapter presents the comprehensive evaluation and analysis of various machine learning algorithms applied to spam email detection. The models explored in this study include the Support Vector Classifier (SVC), Logistic Regression (LR), Decision Tree (DT), Naive Bayes (NB), and BERT. Each algorithm’s performance is assessed based on critical metrics such as accuracy, precision, recall, and F1 score, providing a clear comparison of their effectiveness in identifying spam emails.

the computational environment for the experiments was configured on a machine with an 11th Gen Intel(R) Core(TM) i5-1135G7 processor running at 2.42 GHz, with 8 GB of installed RAM. The device operated on a 64-bit Windows 11 Pro system (version 23H2). This environment facilitated the analysis of the chosen algorithms in this project. where five distinct models were evaluated for their spam detection capabilities.

In this study, TF-IDF was selected as the primary input representation for most models due to its ability to assign importance to terms based on their frequency in the corpus, while also down-weighting common words that appear across many documents. This method enhances the model's ability to differentiate between spam and non-spam content. Although Bag of Words (BoW) was explored, it was only used for a single result to provide a basic comparison. BoW, while simpler, does not account for the significance of terms across documents and often leads to less nuanced representations, making TF-IDF a more robust choice for this analysis.

It is important to note that while traditional models like SVC, LR, and NB use Bag of Words (BoW) or TF-IDF representations for text data, BERT operates differently. BERT does not take BoW or TF-IDF as inputs; instead ,as mentioned in chapter 1, it uses pre-trained embeddings that capture the semantic meaning of words in context. This allows BERT to understand the relationships between words and sentences more effectively, which can significantly enhance its performance in spam detection tasks.

4.1.1 Results on Uncleaned Data Without Hyperparameter Tuning (Imbalanced Dataset)

These results will serve as the **baseline** for comparison with future results, including those obtained from cleaned data, SMOTE and oversampling application, and hyperparameter tuning. This will allow us to understand how different preprocessing and tuning steps impact the performance of various models and which method ultimately performs best in spam email detection.

MODEL	Average Accuracy	Average Precision	Average Recall	Average F1-Score	BEST FOLD
SVC	0.9819	0.9821	0.9819	0.9813	Fold 3
DT	0.9720	0.9635	0.9643	0.9634	Fold 5
LR	0.9720	0.9726	0.9720	0.9707	Fold 3
NB	0.9591	0.9609	0.9591	0.9558	Fold 1
BERT	**0.9905**	**0.983**	**0.976**	**0.9905**	**Fold 1**

Table 4 : Results on Uncleaned Data Without Hyperparameter Tuning (ImbalancedDataset)

BERT achieves an even higher accuracy of **0.9905** and an F1-Score of **0.9905** , outperforming SVC across all metrics. The disparity between the models can be attributed to BERT's advanced ability to understand the context and semantics of words, which contrasts with the traditional text vectorization techniques (such as Bag of Words or TF-IDF) relied upon by other models.

as BERT took approximately **30,877.8 seconds (about 9 hours)** to evaluate due to the complexity of processing pre-trained embeddings and the larger model architecture

Especially when processing text-heavy datasets like email content. This comparison highlights the limitations of traditional machine learning algorithms when faced with more sophisticated deep learning models like BERT, especially in tasks requiring a deep understanding of language and context.

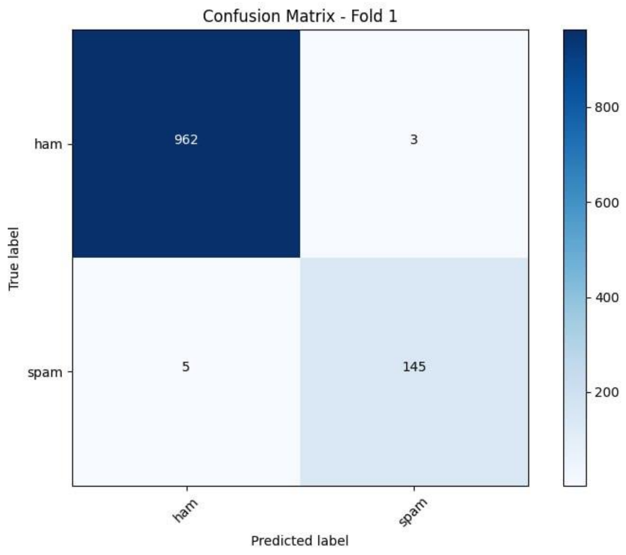


Figure 11 : Confusion Matrix Of BERT Best Fold

The confusion matrix in **figure 11** of Bert best fold. shows that out of 965 "ham" (non-spam) emails, the model correctly identified 962 of them as "ham" while misclassifying 3 as "spam." Similarly, out of 150 "spam" emails, 145 were correctly classified as "spam," but 5 were mistakenly labeled as "ham." The model shows high accuracy with only a few misclassifications, indicating that it effectively distinguishes between spam and non-spam emails. The color gradient indicates the frequency of classifications, with darker shades representing higher values.

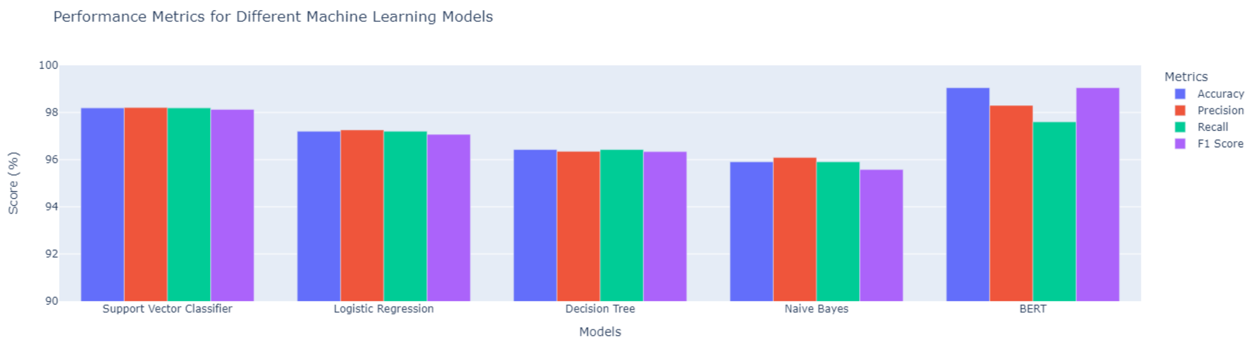


Figure 12 : Comparative Accuracy of Spam Email Detection Models

4.1.2 Results on Cleaned Data Without Hyperparameter Tuning (Imbalanced Dataset)

MODEL	**Average Accuracy**	**Average Precision**	**Average Recall**	**Average F1-Score**	**BEST FOLD**
SVM	0.9718	0.9723	0.9718	0.9703	Fold 2
DT	0.9483	0.9461	0.9483	0.9456	Fold 2
LR	0.9568	0.9577	0.9568	0.9530	Fold 2
NB	0.9540	0.9563	0.9540	0.9492	Fold 2
BERT	**0.9879**	**0.971**	**0.963**	**0.9878**	**Fold 3**

Table 5 : Results on Cleaned Data Without Hyperparameter Tuning(Imbalanced Dataset)

when compared to **BERT**, the performance gap becomes evident. BERT achieves a significantly higher accuracy of **0.9879** and an F1-Score of **0.9878**, far surpassing SVC's performance. as BERT took approximately **29,551.451 seconds (about 8.21 hours)** to evaluate. **BERT** once again demonstrates its superiority, thanks to its ability to handle text data with greater depth and context understanding.

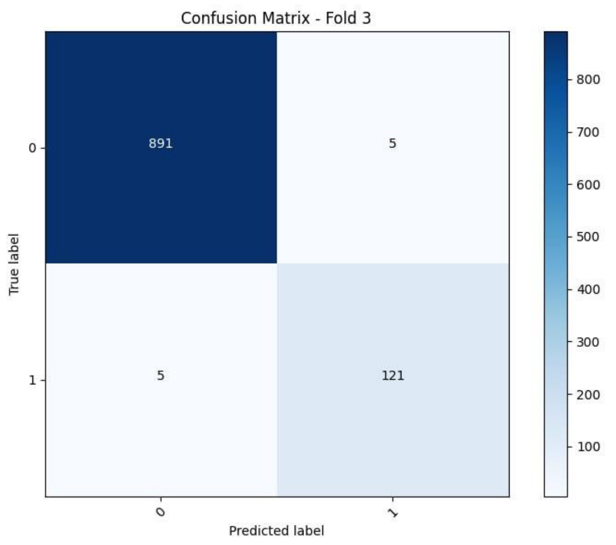


Figure 13 : Confusion Matrix Of BERT Best Fold

The confusion matrix in **figure 13** of Bert best fold. shows that the model correctly identified 891 instances of "ham" (non-spam) out of 896, with 5 being misclassified as "spam." For "spam" emails, the model correctly classified 121 out of 126, with 5 being incorrectly labeled as "ham." The model demonstrates high accuracy, with only minor errors in both the "ham" and "spam" classifications. Similar to the previous matrix.

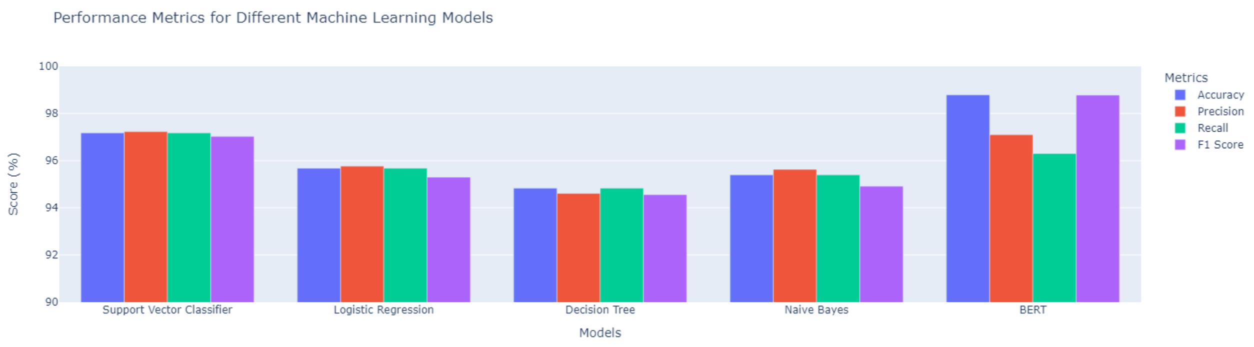


Figure 14 : Comparative Accuracy of Spam Email Detection Models

4.1.3 Results on cleaned data Using Bag of Words (BoW) Without Hyperparameter Tuning (Imbalanced Dataset)

MODEL	**Average Accuracy**	**Average Precision**	**Average Recall**	**Average F1-Score**	**BEST FOLD**
SVC	0.9712	0.9721	0.9712	0.9695	Fold 2
DT	0.9569	0.9559	0.9569	0.9556	Fold 3
LR	0.9748	0.9750	0.9748	0.9736	Fold 3
NB	**0.9804**	**0.9803**	**0.9804**	**0.9800**	**Fold 3**

Table 6 : Results on cleaned data Using Bag of Words (BoW) Without Hyperparameter Tuning(Imbalanced Dataset)

The **best-performing algorithm** in this configuration is **Naive Bayes (NB)**, with the highest accuracy of **0.9804** and an F1-Score of **0.9800**. Despite the theoretical limitations of NB in capturing complex word relationships, its assumptions about word independence seem well-suited to the BoW representation, leading to superior performance.

The BoW model works by converting text into vectors where the frequency of words in the dataset is counted. While effective for simpler models, BoW lacks the ability to capture the semantic relationships between words and the context in which they appear. This can limit its ability to understand the nuances of text.

BERT leverages deep learning and attention mechanisms to capture the meaning of a word in context by looking at the entire sentence rather than relying on word frequency alone. Because of this, **BoW is not applicable to BERT**, while BoW works well with traditional algorithms, it is not suited for modern transformer-based models like BERT.

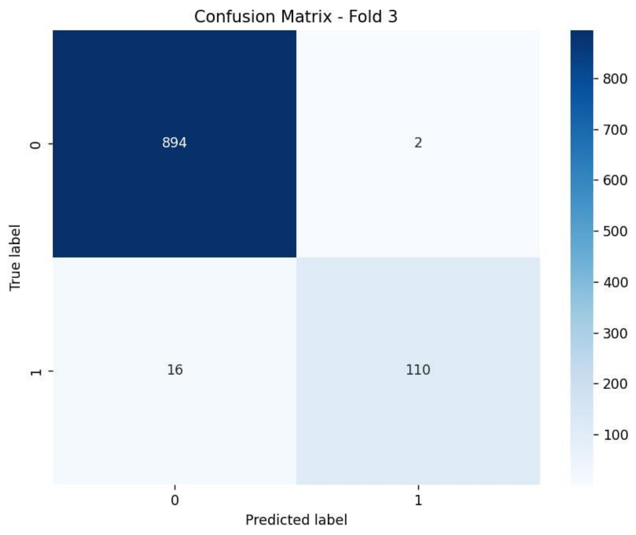


Figure 15 : Confusion Matrix Of LR Best Fold

The Confusion Matrix Of LR Best Fold shown in **figure 15** that out of 896 "ham" (non-spam) emails, the model correctly identified 894 of them as "ham" while misclassifying 2 as "spam." Similarly, out of 126 "spam" emails, 110 were correctly classified as "spam," but 16 were mistakenly labeled as "ham." The model demonstrates high accuracy with minimal misclassifications, effectively distinguishing between spam and non-spam emails, the matrix reflects high accuracy in distinguishing between ham and spam, though with a slightly higher error rate in detecting spam emails.

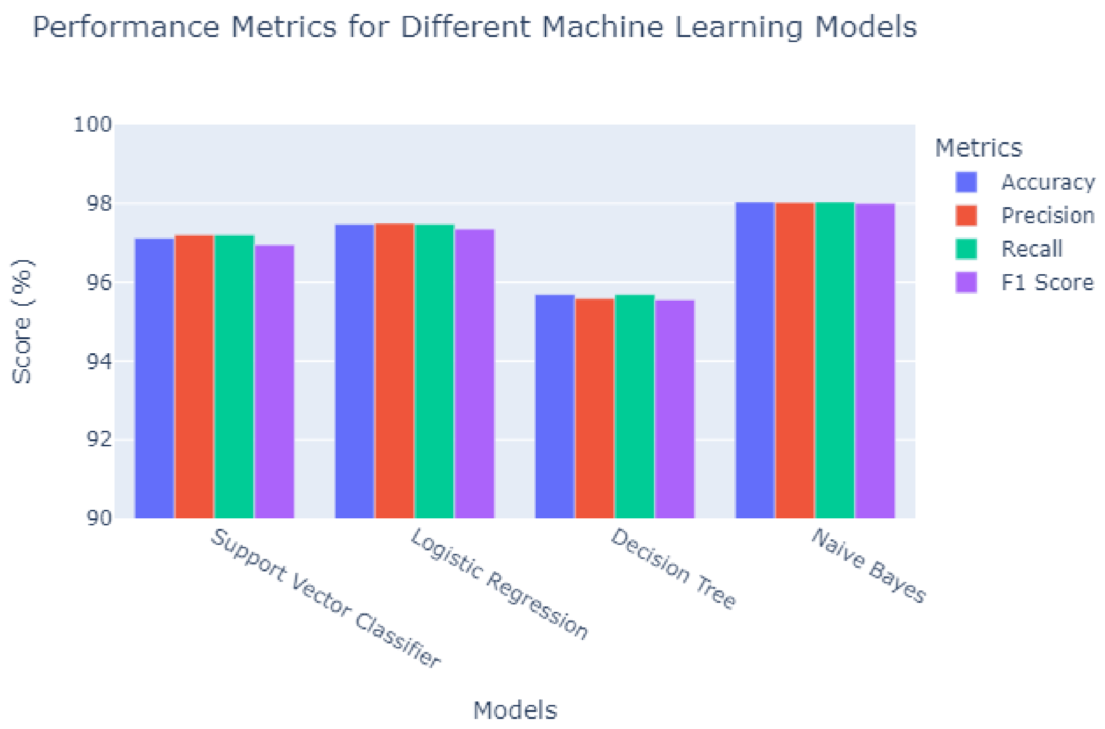


Figure 16 : Comparative Accuracy of Spam Email Detection Models

4.1.4 Results on Cleaned Data With Hyperparameter Tuning (Balanced Dataset)

In this section, the dataset has been balanced using different techniques, BERT processes raw text inputs and transforms them into embeddings based on the context and meaning of words, making it well-suited for text-based **oversampling,** while **SMOTE** was used to balance the dataset for the other algorithms. SMOTE generates synthetic samples by creating numeric vectors based on the feature space of the existing data. The SMOTE implementation utilized the default setting of **5 nearest neighbors** to generate synthetic samples.

hyperparameters for several models were systematically explored to optimize performance. For **BERT**, the tuning focused on learning rate (ranging from *1×10−51 \times 10^{-5}*1×10−5 to *3×10−53 \times 10^{-5}*3×10−5), the number of training epochs (3 to 5), batch size (16 or 32), warmup steps (100, 500, 1000), and weight decay (0.01 to 0.1).

For the **Decision Tree(DT)**, key parameters included the splitting criterion ('gini' or 'entropy'), different splitting strategies ('best' or 'random'), and depth control through `max_depth` values (ranging from None to 50). **Logistic Regression(LR)** was tuned using regularization strength (`C`), penalty types (L1 or L2), and solvers like 'liblinear' and 'saga'. **Support Vector Classifier (SVC)** optimization involved tuning the regularization parameter (`C`), kernel type (linear, polynomial, RBF, or sigmoid), and gamma values. For **Naive Bayes(NB)**, the smoothing parameter (`alpha`) and the option to fit class priors were adjusted to enhance performance.

MODEL	Average Accuracy	Average Precision	Average Recall	Average F1-Score	BEST FOLD
SVC	0.9785	0.9785	0.9785	0.9777	Fold 4
DT	0.9444	0.9422	0.9444	0.9420	Fold 1
LR	0.9808	0.9807	0.9808	0.9805	Fold 3
NB	0.9601	0.9652	0.9601	0.9616	Fold 5
BERT	**0.9888**	**0.981**	**0.966**	**0.9888**	**Fold 3**

Table 7 : Results on Cleaned Data With Hyperparameter Tuning (Balanced Dataset)

BERT achieves an average accuracy of 0.9888 and an F1-Score of 0.9888, surpassing LR’s 0.9808 accuracy and 0.9805 F1-Score. BERT's advanced understanding of language context allows it to better handle complex textual relationships, giving it an edge over traditional models like LR. as BERT took approximately **49,654.9 seconds (about 14 hours)** to evaluate.

although LR excels within traditional models, BERT remains the superior choice for spam classification, demonstrating the advantage of transformer-based architectures in handling text data.

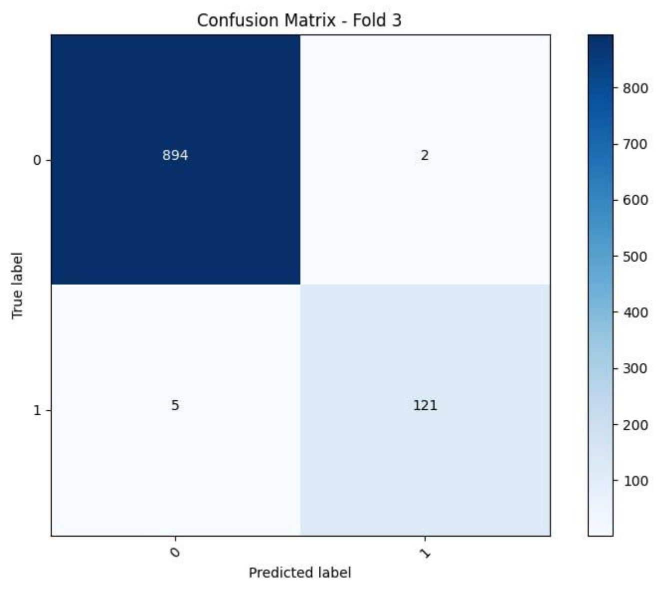


Figure 17 : Confusion Matrix Of BERT Best Fold

The Confusion Matrix Of BERT Best Fold shown in **figure 17** that Out of 896 "ham" (non-spam) emails, the model correctly classified 894 as "ham," with only 2 misclassified as "spam." Similarly, out of 126 "spam" emails, 121 were correctly identified as "spam," while 5 were misclassified as "ham." The model exhibits strong performance with very few misclassifications, effectively distinguishing between spam and non-spam emails. The color gradient highlights the classification frequency, with darker shades indicating a higher count of correctly predicted labels.

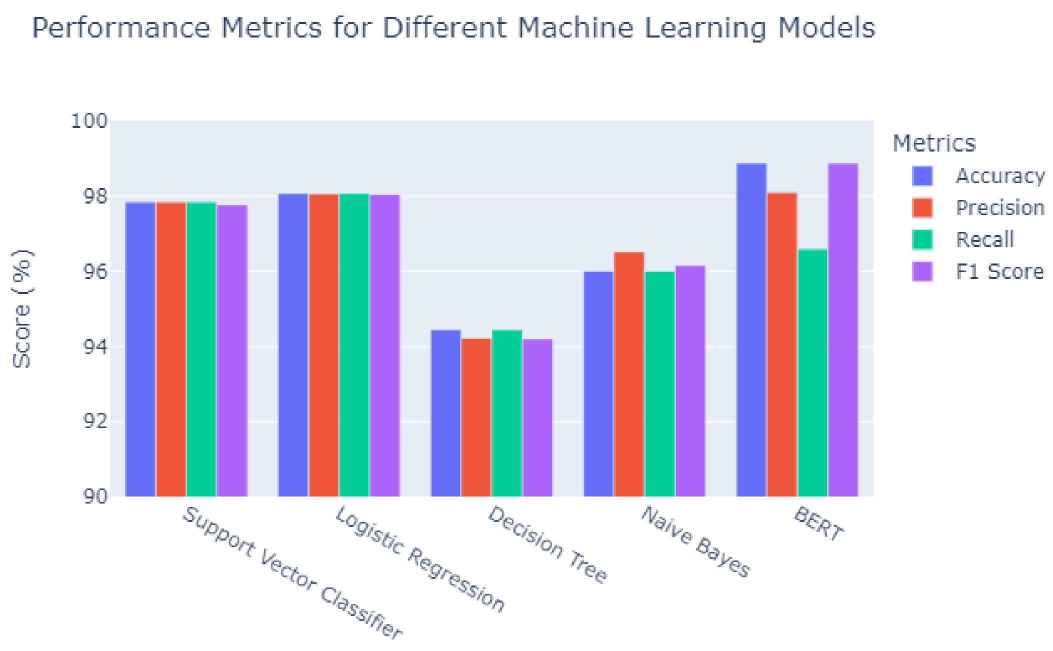


Figure 18 : Comparative Accuracy of Spam Email Detection Models

After evaluating the models and analyzing the results, it became apparent that BERT performed better on uncleaned data compared to the cleaned dataset, achieving higher accuracy. Specifically, when evaluated on uncleaned data, BERT reached 99.05% accuracy, whereas on cleaned data, it attained 98.88%. This suggests that BERT's bidirectional approach, which captures the full context of each word, benefits from retaining the original structure of the messages, including stopwords, symbols, and other elements that traditional preprocessing might remove. These elements seem to provide additional contextual clues that help BERT distinguish between spam and legitimate messages more effectively. This observation confirms that BERT, being a contextually rich model, does not require extensive preprocessing like traditional machine learning algorithms. Its ability to handle raw, unfiltered text allows it to leverage deeper semantic relationships within the data, enhancing its performance in spam detection tasks. The uncleaned model was subsequently integrated into a Streamlit-based website, where it demonstrated effective real-world spam detection capabilities.

Chapter 5

5.1 Introduction to Streamlit Web Application for Spam Detection

The purpose of the Streamlit web application is to provide an accessible and interactive platform for users to classify emails as spam or non-spam (ham) using a pre-trained BERT model. As spam detection is a common task in email filtering, this web application bridges the gap between complex machine learning models and end-users, allowing them to utilize a powerful tool without needing technical expertise.

The application is designed to offer a user-friendly interface where users can easily input email text and receive a classification prediction in real-time. By leveraging Streamlit, a lightweight and intuitive framework for web applications, the system ensures that users can interact with the spam detection model seamlessly. The platform simplifies interaction by automating the preprocessing of user inputs and providing instant results, making it a practical tool for both non-technical users and experts.

The **technical architecture** of the Streamlit application for spam detection integrates several key components, each fulfilling a specific role in ensuring user-friendly interaction and seamless model functionality.

5.2 User Interface Layout and Design

The user interface (UI) design focuses on simplicity and efficiency, making it accessible to users without a technical background. Key functionalities are centralized in one page, minimizing navigation complexities. Each component is intuitively positioned, allowing for a smooth flow from input to output within seconds. The user interface (UI) of the application is minimalistic and designed for ease of use as shown in **Figure 19**, with the following primary elements visible:

- **Text Area**: This input field allows users to enter the email or message they want to classify as spam or ham (not spam). It is clearly labeled to guide users in providing the necessary input.
- **Classify Button**: A large button labeled or "Submit" triggers the backend model to process the entered message. This button is essential for user interaction and runs the BERT model for classification.
- **Classification Output**: After processing, the application presents the result (either "Spam" or "Ham") in a text box or bolded label format. The design ensures immediate visibility of the result, making the process straightforward

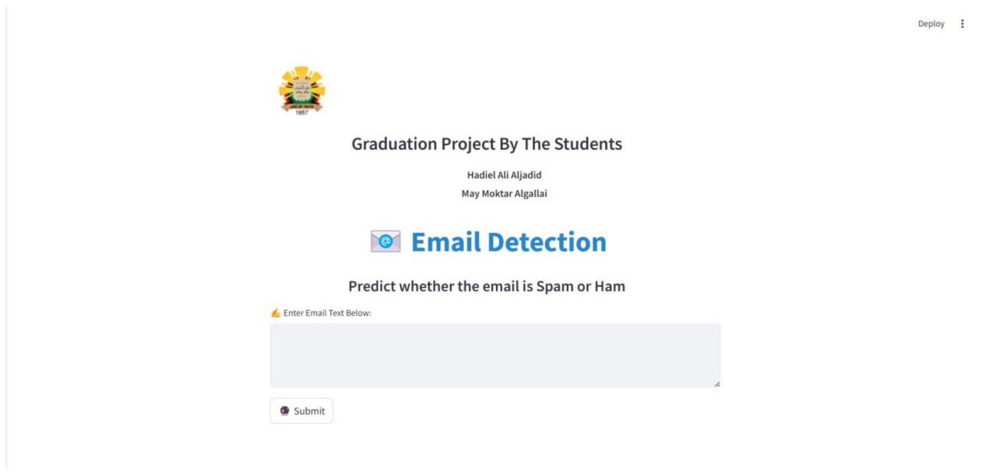


Figure 19 : Streamlit Web Application User Interface Layout

5.3 Model Integration

For the model integration in the Streamlit application, the **uncleaned data BERT model**[3] was selected due to its superior performance in handling raw, unfiltered text. As mentioned in Chapter 4, the BERT model achieved 99.05% accuracy on uncleaned data, compared to 98.88% on cleaned data. This performance difference highlights BERT's ability to capture deeper contextual relationships from raw text, including stopwords, symbols, and other elements often removed during traditional preprocessing.

[3] https://github.com/mayalgallai/hadiel-Al-jadid-graduation-project-

The model integrated into the application is the version with no hyperparameter tuning and no oversampling, aligning with the highest accuracy achieved during model evaluation. The uncleaned model was chosen specifically to leverage BERT's strengths in processing raw email content, making it well-suited for real-world spam detection tasks. The integration was handled as follows:

- **Loading the Pre-Trained Model:** The BERT model and tokenizer are loaded from a pre-trained model checkpoint using the Hugging Face Transformers library. This allows the application to process inputs through BERT's attention-based architecture and return predictions with high accuracy.
- **Processing User Input:** Once a message is input by the user, it is tokenized using BERT's tokenizer, which converts the raw text into input tokens that BERT can process. The tokenized input is then passed to the BERT model for prediction.
- **Making Predictions:** The model outputs a probability distribution indicating whether the message is classified as spam or ham. Based on this prediction, the result is displayed back to the user, providing instant feedback.

5.4 User Interaction

The Streamlit web application provides a straightforward, user-friendly interface that allows users to easily interact with the BERT model for spam detection. The interaction process is designed to be intuitive and requires minimal effort from the user. Below is a step-by-step demonstration of how users can interact with the application, referencing screenshots to illustrate the process.

- **Inputting the Email Text**

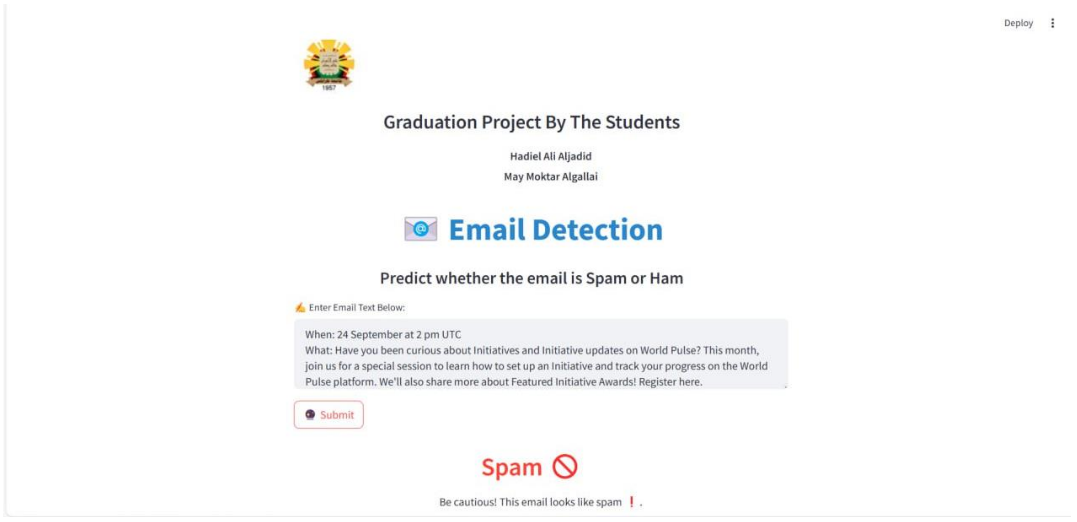
The application presents a clean and simple text area, clearly labeled for users to input their email message as shown in **Figure 19**. Users are prompted to enter the content of the email they wish to classify as either spam or ham (non-spam). The text area is designed to handle both short and long email messages without overwhelming the user with unnecessary fields or options.

- **Submitting the Input**

  Once the user has entered their email text, they can proceed by clicking the "Submit" button. This button is prominently displayed and activates the backend BERT model to process the input message. The "Submit" button is the core interaction point, allowing the user to trigger the model's prediction with a single click.

- **Receiving the Prediction**

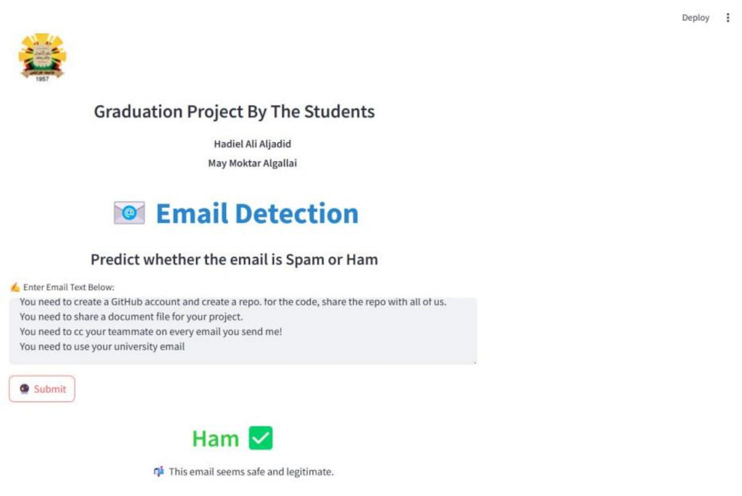
  After the user clicks "Submit," the BERT model processes the input and displays the result on the same page, as shown in **Figure 20 / 21**. The result is either "Spam" or "Ham," clearly visible in a text box or bold label format. This immediate feedback ensures that the user can quickly determine whether the email is spam or legitimate (ham), without needing to refresh or navigate to a new page.



When: 24 September at 2 pm UTC
What: Have you been curious about Initiatives and Initiative updates on World Pulse? This month, join us for a special session to learn how to set up an Initiative and track your progress on the World Pulse platform. We'll also share more about Featured Initiative Awards! Register here.

Be cautious! This email looks like spam ❗.

Figure 20 : Spam classification result displayed in the Streamlit app.


Deploy
Graduation Project By The Students
Hadiel Ali Aljadid
May Moktar Algallai
Email Detection
Predict whether the email is Spam or Ham
Enter Email Text Below:
Submit
Ham ✓
This email seems safe and legitimate.

You need to create a GitHub account and create a repo. for the code, share the repo with all of us.
You need to share a document file for your project.
You need to cc your teammate on every email you send me!
You need to use your university email

Figure 21 : ham classification result displayed in the Streamlit app.

- **Interpreting the Results**

The classification result, displayed in bold near the input area, is simple to interpret. If the result is "Spam," the user is informed that the email is likely to be unsolicited or harmful. If the result is "Ham," it suggests that the email is legitimate. The clear presentation of results minimizes confusion and makes the process accessible to users of all technical levels.

These steps showcase the smooth user experience provided by the Streamlit application. The simplicity of the interface, combined with the powerful underlying BERT model, ensures that users can effectively classify emails with minimal interaction.

5.5 Limitations and Future Work

Despite the strengths of the Streamlit application, there are some limitations that must be acknowledged. One major limitation is computational constraints. Running a BERT model in real time, even without hyperparameter tuning or extensive preprocessing, can be resource-intensive, especially for users with limited computational power. This may result in slower response times or the inability to process larger datasets efficiently. Additionally, the user interface, while functional, could benefit from more advanced features, such as customizable input formats or options for batch processing, which would enhance usability for a wider range of users.

In terms of future work, several improvements could be explored. Enhancing the application’s performance by optimizing model loading times and reducing memory usage would make it more accessible to users with basic hardware. The integration of additional features, such as multi-language support or real-time feedback on model confidence scores, would further improve the user experience. Furthermore, expanding the model’s capabilities to handle other forms of textual data, such as social media posts or instant messages, would broaden its practical applications.

One promising direction for future research is **Cross-Domain Spam Detection**, which involves adapting spam detection models to operate effectively across different platforms, such as social media, instant messaging, and other communication channels. Investigating how well the BERT model generalizes to these domains would provide valuable insights into its versatility and potential improvements in spam filtering systems. Another area of future exploration is analyzing the **correlation of words** within spam messages across various domains. Understanding how specific word patterns contribute to spam classification in different contexts could lead to the development of more robust and accurate spam detection models.

References

Abadi et al. (2016). *TensorFlow: A system for large-scale machine learning. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 265-283.*

al, B. e. (2009). *Natural Language Processing with Python. O'Reilly Media, Inc.*

Androutsopoulos, I. K. (2000). *"An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages." Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval. ACM.*

Arlot, S. &. (2010). *"Cross-Validation: A Brief Review".*

Arora, S., Sharma, S., & Verma, P. (2020). *BERT based spam detection approach: An application to improve email security. IEEE Access.*

Bird, S. (2006). *NLTK: The Natural Language Toolkit.*

Brownlee. (2018). *How to Configure the Learning Rate Hyperparameter When Training Deep Learning Neural Networks. Machine Learning Mastery.*

Brownlee, J. (. (n.d.). *How to Improve Performance for Imbalanced Classification. Machine Learning Mastery.*

Chawla, e. a. (2002). *SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321-357.*

Cisco. (2020). *Cisco. (2020). Cisco Annual Internet Report (2018–2023). Cisco Systems, Inc.*

Cortes & Vapnik. (1995). *Support-vector networks. Machine Learning.*

Devlin et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.*

Ditzler, G., Morrison, J., & Lan, Y. (2015). *A methodology for logistics regression-based spam filtering. Journal of Machine Learning Research, 16(1).*

FBI. (2020). *FBI. (2020). Internet Crime Report 2019. Federal Bureau of Investigation.*

Foundation, P. S. (2024). *re — Regular expression operations. Python Documentation. Retrieved from https://docs.python.org/3/library/re.html.*

Foundation., P. S. (n.d.). *os — Miscellaneous operating system interfaces. Python 3.8.6 documentation.* Retrieved from 2024.

G. V. Cormack. (2007). *"Email spam filtering: A systematic review," Foundations and Trends in Information Retrieval, vol. 1, no. 4, pp. 335-455 .*

Géron, A. (. (n.d.). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.). O'Reilly Media. (Covers Grid Search and Random Search in the context of hyperparameter tuning.).*

Goodfellow, e. a. (2016). *Deep Learning.*

Guo, X. Y. (2020). *"On the performance of hybrid SVM for spam detection: Feature selection and ensemble techniques." Expert Systems with Applications,.*

He, e. a. (2009). *Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering.*

Heimerl et al. (2014). *Word cloud explorer: Text analytics based on word clouds. In 2014 47th Hawaii International Conference on System Sciences (pp. 1833-1842). IEEE.*

Hosmer et al. (2013). *Applied Logistic Regression (3rd ed.). Wiley.*

Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering.*

Joachims, T. (1998). *Text categorization with support vector machines: Learning with many relevant features. In Proceedings of the European Conference on Machine Learning (ECML '98),.*

Kohavi, R. (1995). *"A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection".* International Joint Conference on Artificial Intelligence (IJCAI).

Kohavi, R. (1995). *A study of cross-validation and bootstrap for accuracy estimation and model selection.*

Lemaître et al. (2017). *Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. Journal of Machine Learning Research,.*

Li, J. L. (2019). *"Enhancing the effectiveness of logistic regression models for spam detection via feature selection." International Journal of Information Technology & Decision Making,.*

Loper, et al. (2002). *NLTK Book.*

Manning, e. a. (2008). *Introduction to Information Retrieval. Cambridge University Press.*

McKinney, W. (2010). *Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference.*

Metsis et al. (2006). *Spam Filtering with Naive Bayes - Which Naive Bayes? CEAS.*

Meyer, D. L. (2004). *"The support vector machine under test." Neurocomputing.*

Mikolov, T. C. (2013). *Efficient Estimation of Word Representations in Vector Space.*

Millman, K. J. (2011). *Python for Scientists and Engineers. Computing in Science & Engineering, .*

Paszke et al. (2019). *PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems (NeurIPS 2019).*

Pedregosa, F. V. (2011). *Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research.*

Plotly Technologies INC. (2015). *Plotly Graphing Library Documentation.*

Python Software Foundation. (2024). *abc — Abstract Base Classes. Python 3.8.6 documentation.*

Quinlan, J. R. (1986). *Induction of decision trees. Machine Learning,.*

Radicati. (2020-2024). *Radicati, S. (2020). Email Statistics Report, 2020-2024. The Radicati Group, Inc.*

Ramos, J. (2003). *Using TF-IDF to Determine Word Relevance in Document Queries. In Proceedings of the First International Conference on Machine Learning and Cybernetics .*

Sahami, e. a. (1998). *A Bayesian Approach to Filtering Junk E-Mail. In AAAI Workshop on Learning for Text Categorization.*

Sahami, M. e. (1998). *"A Bayesian approach to filtering junk e-mail." Learning for Text Categorization: Papers from the 1998 Workshop.*

Sculley et al. (2009). *"Spam filtering with machine learning in the new anti-spam generation." Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Sculley. et al. (2007). *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval.the importance of optimizing hyperparametersin SVMs for spam filtering, particular in balancingprecision and recall.*

Sokolova, e. a. (2009). *A systematic analysis of performance measures for classification tasks. Information Processing & Management.*

Statista, G. a. (2023). *"Email Spam Statistics.*

Symantec. (2019). *Internet Security Threat Report. Volume 24.*

Times, T. N. (2016). *Russian Hackers Used Phishing to Hack Democratic National Committee Emails.*

Van der Walt, S. C. (2011). *The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering.*

Vaswani, A. S. (2017). *Attention is all you need. Advances in Neural Information Processing Systems.*

Verizon, E. (2020). *Data Breach Investigations Report (DBIR).*

Wang, Y. M. (2003). *Decision tree methods for spam email detection. In Proceedings of the 4th International Conference on Computer Science and its Applications (ICCSA '03).*

Yang, Y., & Pedersen, J. O. (1997). *A comparative study on feature selection in text categorization. In Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97).*

Zeeshan Younas, Muhammad Ibrahim Qasmi, Sheikh Muhammad Abdullah. (n.d.). *Dataset Notebooks*. Retrieved from https://www.kaggle.com/datasets/zeeshanyounas001/email-spam-detection/code

Zhang, Z. L. (2022). *"Spam email detection using a decision tree approach enhanced with word embeddings." Journal of Information Security and Applications.*