Spam Message Classification using Naive Bayes classifier and BERT-based NLP model

Submitted by:

Name: Sadhana S

Register Number: 125018057

Under the guidance of

Dr. Swetha Varadarajan

<u>Table of contents</u>

Abstract

In natural language processing (NLP), spam detection is an essential activity that guarantees the effectiveness and security of communication networks.

The efficiency of the Multinomial, Bernoulli, and Gaussian Naive Bayes classifiers in identifying spam from non-spam communications is investigated in this study. Each model's performance is assessed using metrics including accuracy score, precision score, and confusion matrix using a labeled dataset of messages. This study's first part compares the three Naive Bayes variations to identify the best spam detection model.

The project is further extended by comparing its performance to conventional Naive Bayes classifiers using a sophisticated NLP-based technique, like a Transformer model or BERT.
This comparative analysis provides insights into the trade-offs between classical machine learning models and state-of-the-art NLP models for effective spam detection.

Introduction

Digital communication networks face a great challenge from spam communications, which inundate consumers with unsolicited and frequently malicious content. Ensuring that communications classified as spam or ham are properly distinguished from one another is crucial to preserving the integrity of communication channels.

This project aims to create and assess machine learning models for classifying spam messages using three different Naive Bayes classifier variants: Multinomial, Bernoulli, and Gaussian.

This study has two main goals: (i) to evaluate how well each Naive Bayes model detects spam messages, and (ii) to broaden the analysis by adding a more sophisticated NLP-based model, like a Transformer-based model or BERT, to offer a comparison of old versus new methods.

Through this, the project aims to provide comprehensive insights into the trade-offs between classical machine learning techniques and state-of-the-art NLP methods in the context of spam detection.

The goal of this research is to create a reliable and accurate classifier that can distinguish between messages that are spam and those that are not.

To do this, a binary classification problem must be created, with the objective being to classify each message as "spam" or "ham". To determine which models are most appropriate for use in spam detection in the real world, their performance will be assessed using common metrics including *accuracy score, and precision score.*

Related Work

A base paper was chosen for the initial implementation using Naive Bayes Algorithm. Implementation of Spam Classifier using Naïve Bayes Algorithm (Link to the Research article).
This paper was used only to build up an initial blueprint.
The spam classification dataset was downloaded from Kaggle. It is originally available in the UCI Machine Learning repository as well. Code to Dataset

Background Work and Methodology

1. Dataset:

(Kaggle/UCI) - https://archive.ics.uci.edu/dataset/228/sms+spam+collection

The dataset used for this project comprises labeled messages categorized as spam or ham. The data is pre-processed to ensure quality and relevance for training and evaluation.

The dataset undergoes an extensive exploratory data analysis (EDA) to understand the distribution, common patterns, and key features distinguishing spam messages from non-spam ones.
The dataset has 5572 instances and 2 columns of features.

2. Data Cleaning and Preprocessing:

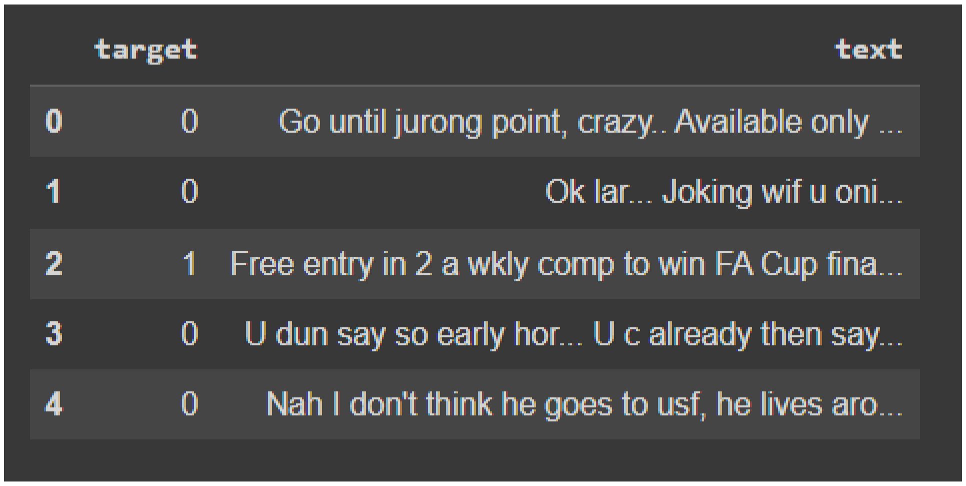
Before model training, the data undergoes a thorough cleaning and preprocessing steps to prepare it for model building:

- Tokenization: Each message is broken down into individual tokens (words).

- Stop Word Removal: Commonly occurring words that do not contribute significantly to the model's predictive power, such as "is," "and," and "the," are removed.
- Stemming: Words are reduced to their base or root form to ensure consistency and minimize dimensionality

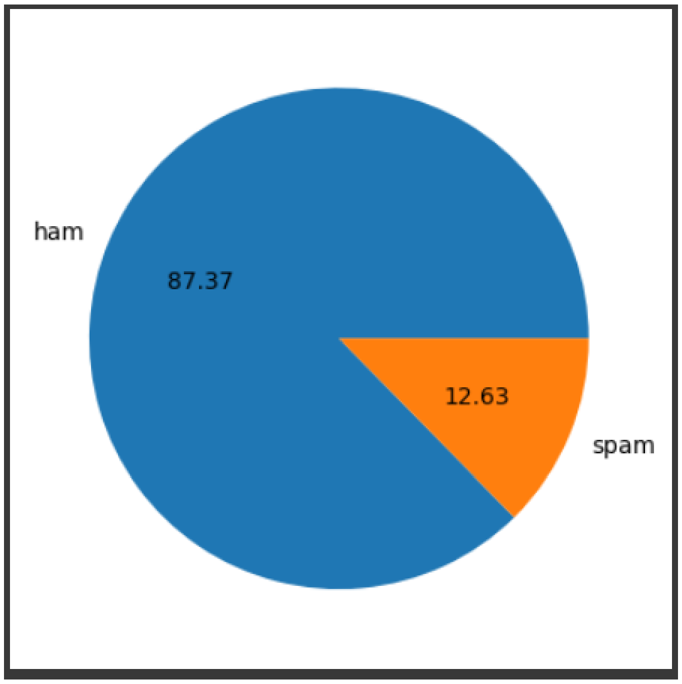
These preprocessing steps help normalize the text data and reduce noise, allowing the models to learn more effectively from the meaningful content.

OUTPUT:

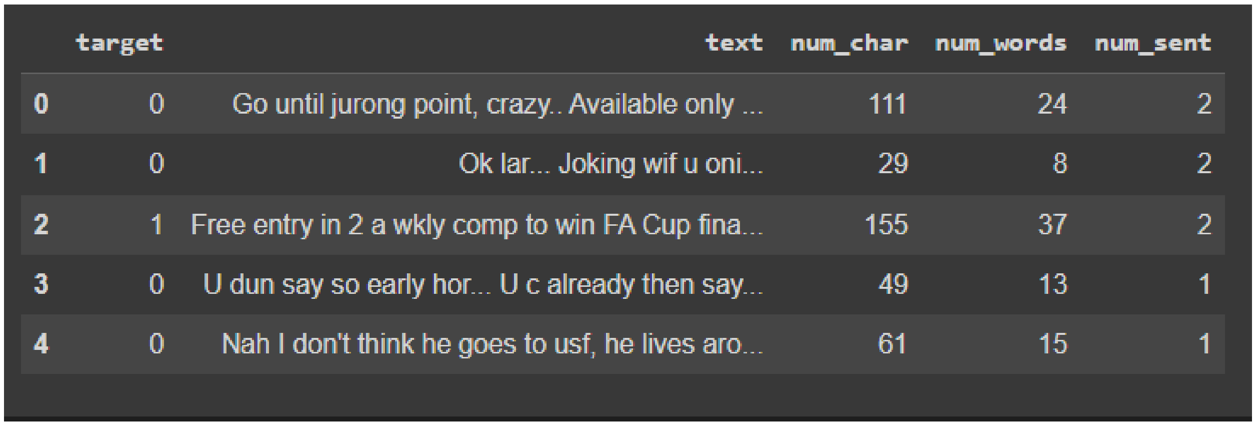


	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

Dataset after one hot encoding of the target column



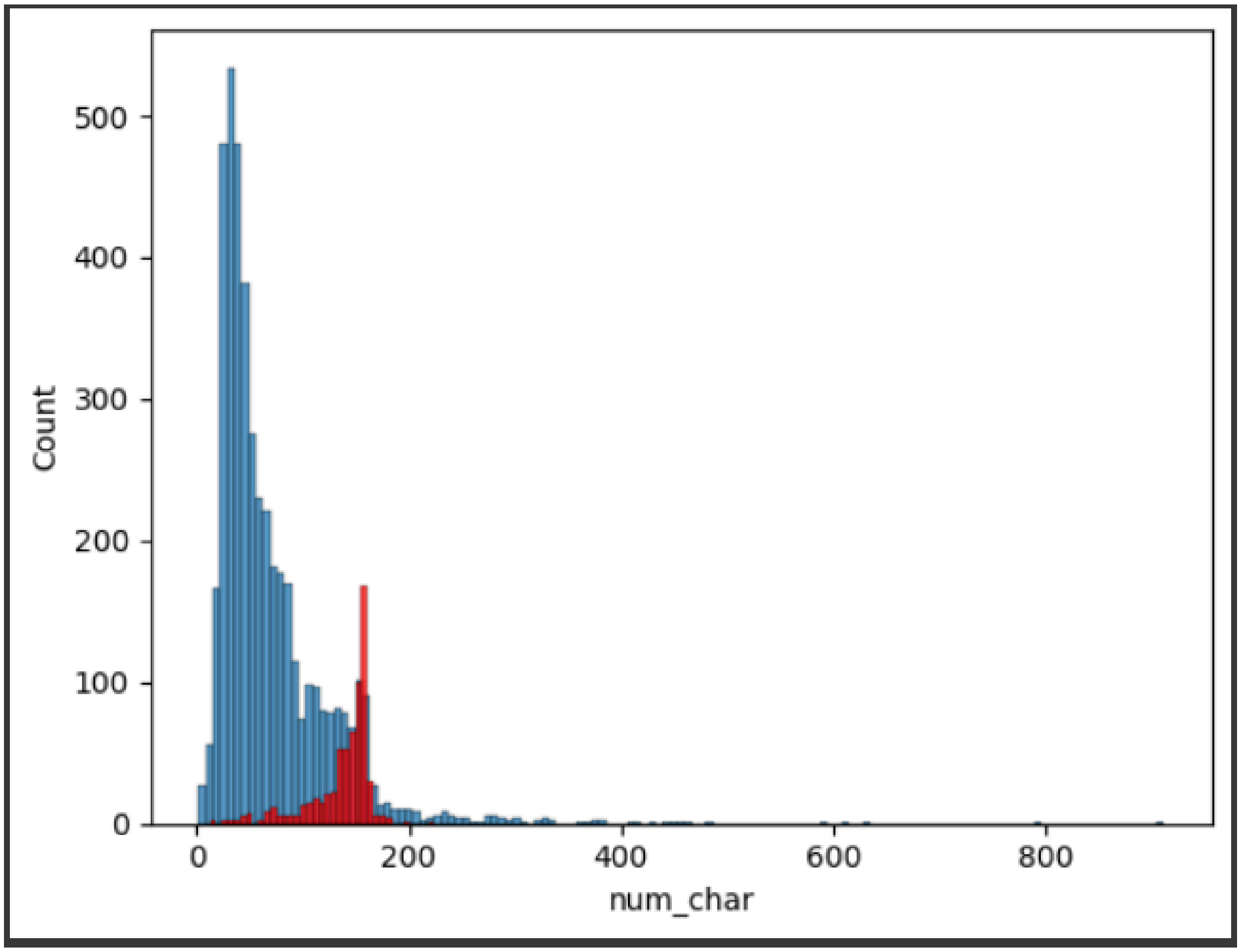
Pie chart showcasing the imbalance of the dataset

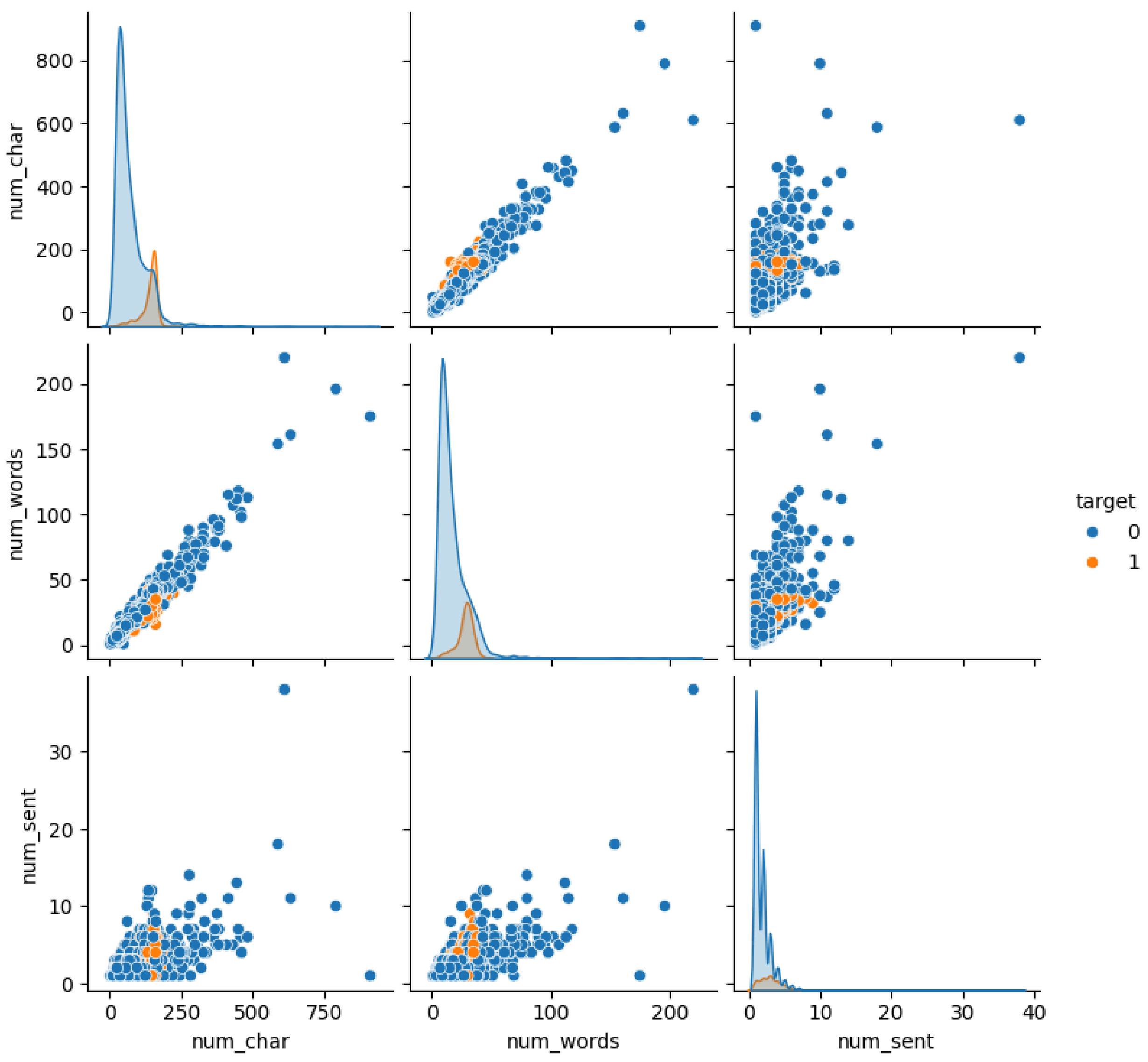


	target	text	num_char	num_words	num_sent
0	0	Go until jurong point, crazy.. Available only ...	111	24	2
1	0	Ok lar... Joking wif u oni...	29	8	2
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2
3	0	U dun say so early hor... U c already then say...	49	13	1
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1

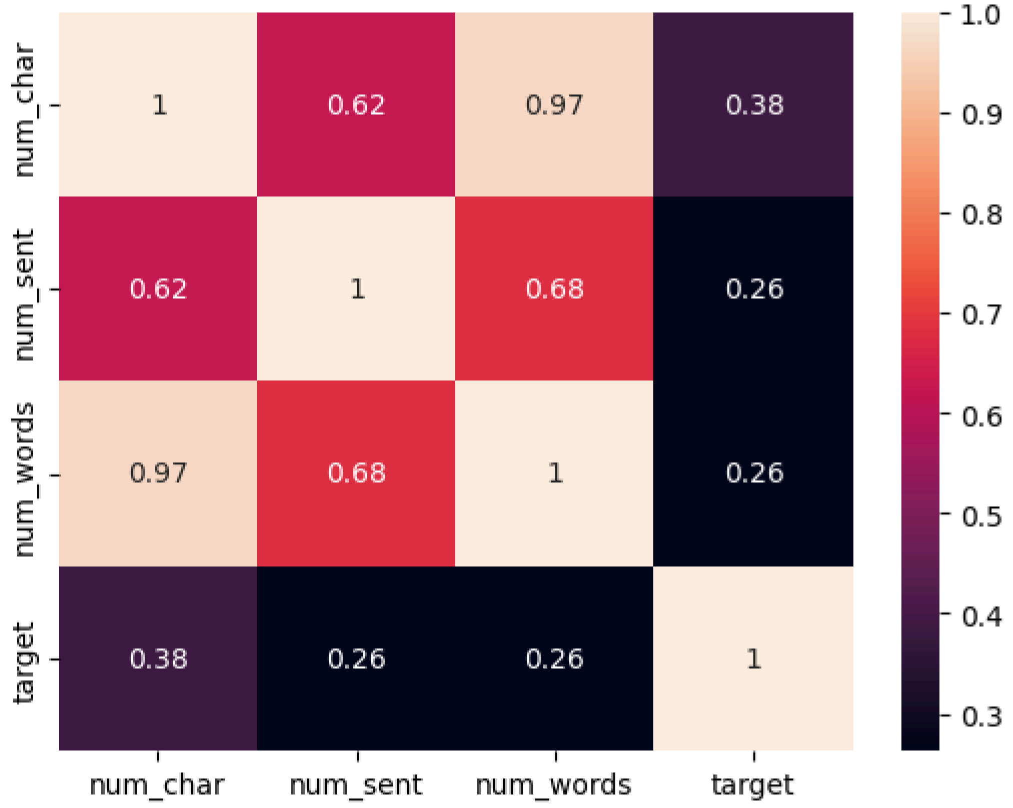
Number of characters, words, and sentences studied for both types of messages.

EDA on the Dataset:

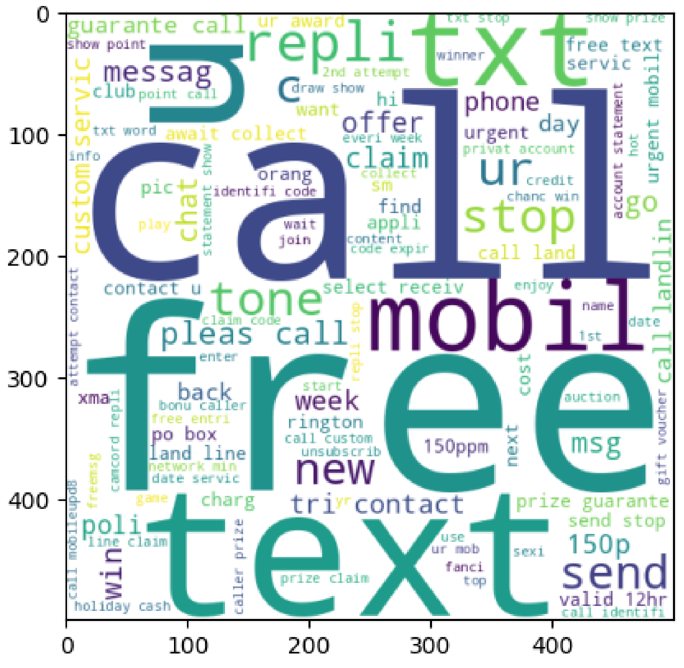




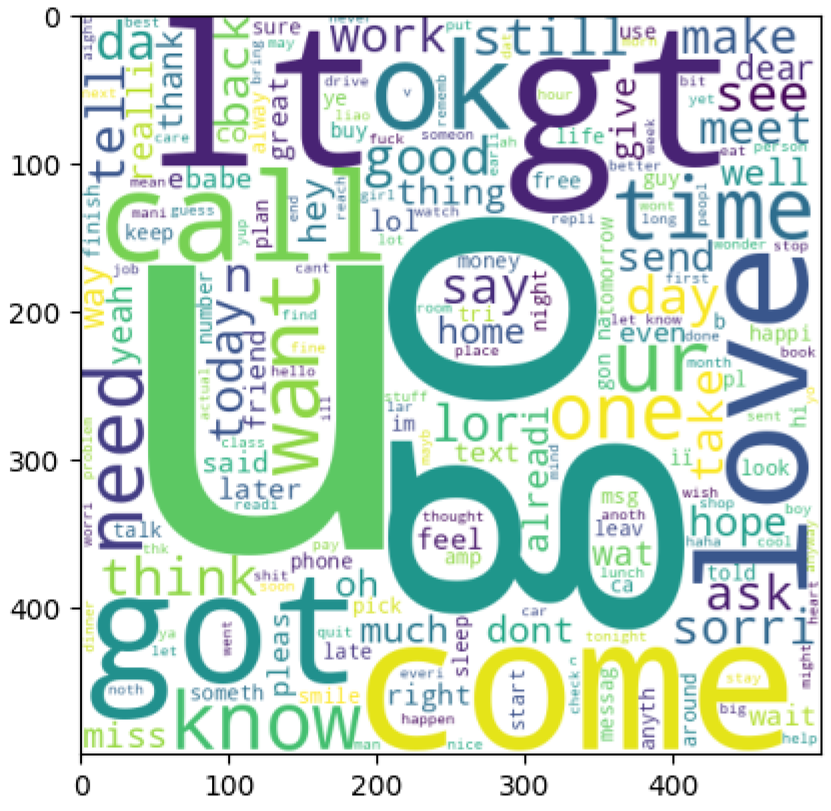
num_char
num_words
num_sent
target
0
1



Correlation heatmap that signifies that the number of characters is strongly correlated with the target



Wordcloud of Spam messages



Wordcloud for Ham Messages

3. Machine Learning Models:
The project focuses on three variants of Naive Bayes classifiers:

- Multinomial Naive Bayes: Typically used for text classification problems with discrete features (word counts). It assumes that the features follow a multinomial distribution.
- Bernoulli Naive Bayes: Suitable for binary/boolean features, this variant considers the presence or absence of words rather than their frequency.
- Gaussian Naive Bayes: Assumes that the features follow a Gaussian (normal) distribution, though it is less commonly used for text classification.

Types of Naive Bayes:
Bernoulli Naive Bayes
Bernoulli Naive Bayes is a variation of Naive Bayes used for **binary/boolean** data. In this model, features are binary-valued, which means each feature can take on only two values: **1** (present) or **0** (absent). It is commonly applied in text classification tasks where the model checks whether a particular word appears in a document or not.

How It Works:

- **Feature Representation**: The input data for Bernoulli Naive Bayes is typically represented in a binary format. For example, in text classification, each document is converted into a binary vector where each element indicates the presence (1) or absence (0) of a specific word.

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

- **Bayes' Theorem**: Like other Naive Bayes classifiers, Bernoulli Naive Bayes relies on Bayes' Theorem. Given a document, the model computes the probability of the document belonging to a particular class (e.g., spam or not spam) based on the presence or absence of words.
- **Assumption of Conditional Independence**: Like all Naive Bayes models, Bernoulli Naive Bayes assumes that the presence or absence of each word is **independent** of other words in the document.
- **Binary Feature Model**: Bernoulli Naive Bayes models the presence/absence of features rather than their frequency (unlike Multinomial Naive Bayes, which considers word counts). This model calculates two probabilities for each feature (word) in each class:
    - **P(word present | class)**
    - **P(word absent | class)**

Strengths of Bernoulli Naive Bayes:

- **Efficient for binary features**: Works well for binary data where only the presence or absence of a feature matters. This makes it ideal for text classification tasks like spam detection or sentiment analysis.
- **Handles small datasets well**: Can achieve good results even with limited training data.
- **Performs well with feature selection**: Works particularly well when certain features (words) are more informative in classification tasks.

Limitations of Bernoulli Naive Bayes:

- **Over-simplified assumptions**: Assumes that features (e.g., the presence of words) are conditionally independent, which may not be true in real-world data.

- **Binary features only**: It only considers whether a feature (word) is present or absent. It does not take into account the frequency of the word, which could sometimes be important in text analysis.
- **Zero-frequency problem**: If a word (feature) never appeared in the training set for a given class, the model assigns a probability of zero to documents containing that word. This issue can be addressed using **smoothing techniques** like Laplace smoothing.

Use Case: Bernoulli Naive Bayes is particularly effective in tasks where binary feature representation is most appropriate. For example:

- **Spam classification**: Where the presence or absence of certain keywords (e.g., "win," "free") helps classify an email as spam or not.
- **Document classification**: Useful in text classification tasks where binary occurrences of words are considered more important than their frequencies.

Gaussian Naive Bayes

Gaussian Naive Bayes is a variant of Naive Bayes designed for continuous data. It assumes that each feature follows a normal (Gaussian) distribution for each class. This makes it suitable for tasks with real-valued inputs, such as measurements or sensor data.

How It Works:

- **Normal Distribution Assumption**: For every feature in the data, Gaussian Naive Bayes assumes it follows a bell-curve distribution within each class.
- **Calculating Probabilities**: The classifier computes the likelihood of a data point belonging to a particular class based on the values of its features. For example, in a fruit classification task, it would calculate the probability that a given fruit's weight and size match those typically seen in apples or oranges.
- **Conditional Independence**: Like other Naive Bayes models, Gaussian Naive Bayes assumes that all features are independent given the class label, meaning the value of one feature doesn't affect the others.

Strengths:

- **Fast and efficient**: It's quick to train and make predictions, making it ideal for real-time applications.
- **Works well with small data**: Even with limited training data, Gaussian Naive Bayes can perform effectively.

- **Handles continuous features**: It's well-suited for tasks where features are real-valued (e.g., height, weight, temperature).

Limitations:

- **Normality assumption**: It assumes that the features are normally distributed, which may not be true in many real-world cases. If the data doesn't follow this distribution, the model's accuracy may suffer.
- **Independence assumption**: The model assumes that the features are independent of each other, which is often unrealistic since real-world features can be correlated.

Use Cases:

- **Medical diagnosis**: Predicting diseases based on continuous features like heart rate or blood pressure.
- **Financial risk prediction**: Estimating risk using continuous indicators such as stock prices or interest rates.
- **Sensor data analysis**: Analyzing data from sensors in devices or environments, like temperature or humidity readings.

Multinomial Naive Bayes

Multinomial Naive Bayes is a variant of Naive Bayes that is well-suited for **discrete, count-based data**. It is commonly used in tasks where the features represent frequencies or counts, such as text classification problems (e.g., spam detection or document categorization).

How It Works:

- **Count-based Features**: Multinomial Naive Bayes works with data that represents counts, such as the number of times a word appears in a document. It calculates the likelihood of a class by looking at how frequently different features (like words) occur in a given class.
- **Conditional Probabilities**: For each class, the model calculates the probability of each feature (e.g., each word in a text) occurring. The classifier then multiplies these probabilities to determine the overall likelihood that the data point (e.g., a document) belongs to each class.
- **Conditional Independence**: Like all Naive Bayes classifiers, it assumes that features are independent given the class label, meaning the occurrence of one word doesn't influence the occurrence of another.

Strengths:

- **Effective for text data**: It's highly effective for text classification tasks where the data consists of word counts, such as spam filtering or sentiment analysis.
- **Fast and efficient**: Multinomial Naive Bayes is simple and computationally efficient, making it suitable for large-scale problems with high-dimensional data (like thousands of words).
- **Performs well with small datasets**: It can perform well even with limited amounts of data, which is especially useful for tasks where gathering large datasets is challenging.

Limitations:

- **Requires count-based data**: Multinomial Naive Bayes assumes that features are discrete and represent counts. It is not suitable for continuous features like height or temperature unless those features are discretized.
- **Independence assumption**: As with all Naive Bayes classifiers, it assumes that the features are independent given the class, which may not be true in many real-world applications (e.g., words in a sentence may be correlated).

Use Cases:

- **Text classification**: Multinomial Naive Bayes is frequently used in Natural Language Processing (NLP) tasks such as spam detection, document classification, and sentiment analysis.
- **Document categorization**: Grouping documents based on topics or identifying the category of news articles based on word frequencies.

These models will be trained on the preprocessed data, and their performance will be evaluated using metrics like *accuracy score, precision score, and confusion matrix.*

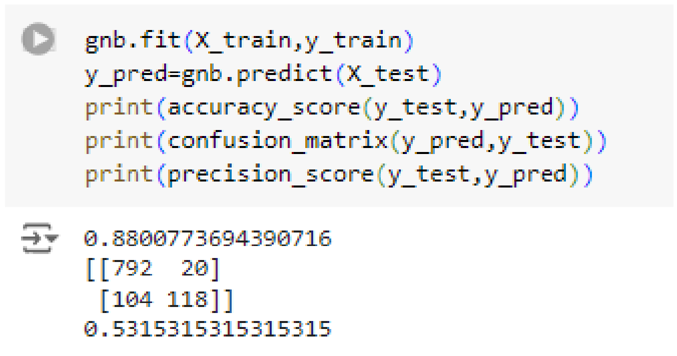
4. Model Building:

The preprocessed data was split into train and test datasets with a ratio of 80:20, and the data was trained with the three different Naive Bayes models.

The processed texts were vectorized using two different vectorizers: Count Vectorizer and TFIDF Vectorizer.
It was observed that the TFIDF vectorizer did a better job because it assigns a weight for each word based on its frequency.

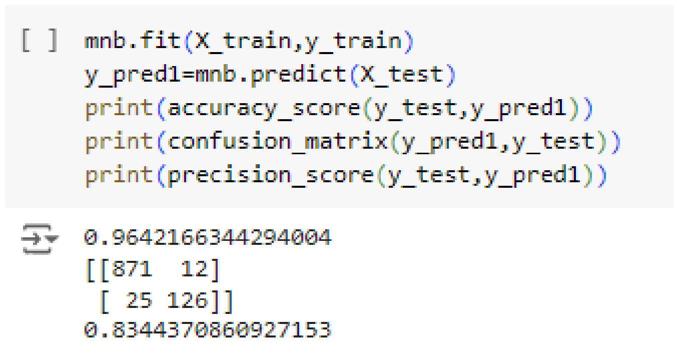
OBSERVATIONS:
Multinomial Naive Bayes had a perfect precision score of 100. Therefore that model was chosen.

COUNT VECTORIZER:



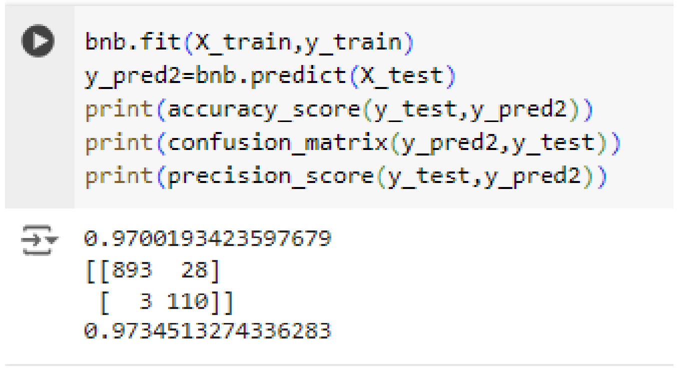
```
gnb.fit(X_train,y_train)
y_pred=gnb.predict(X_test)
print(accuracy_score(y_test,y_pred))
print(confusion_matrix(y_pred,y_test))
print(precision_score(y_test,y_pred))
```

```
0.8800773694390716
[[792  20]
 [104 118]]
0.5315315315315315
```



```
mnb.fit(X_train,y_train)
y_pred1=mnb.predict(X_test)
print(accuracy_score(y_test,y_pred1))
print(confusion_matrix(y_pred1,y_test))
print(precision_score(y_test,y_pred1))
```

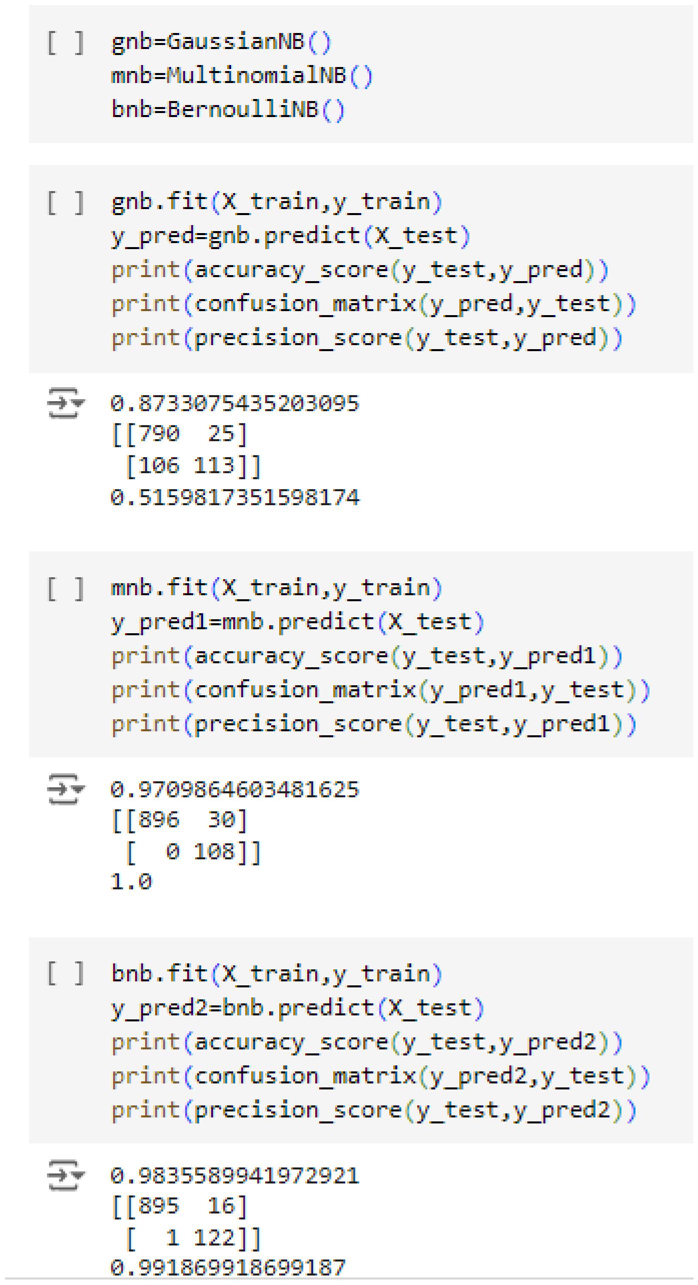
```
0.9642166344294004
[[871  12]
 [ 25 126]]
0.8344370860927153
```



```
bnb.fit(X_train,y_train)
y_pred2=bnb.predict(X_test)
print(accuracy_score(y_test,y_pred2))
print(confusion_matrix(y_pred2,y_test))
print(precision_score(y_test,y_pred2))
```

```
0.9700193423597679
[[893  28]
 [  3 110]]
0.9734513274336283
```

TFIDF VECTORIZER:



```
[ ] gnb=GaussianNB()
    mnb=MultinomialNB()
    bnb=BernoulliNB()
```

```
[ ] gnb.fit(X_train,y_train)
    y_pred=gnb.predict(X_test)
    print(accuracy_score(y_test,y_pred))
    print(confusion_matrix(y_pred,y_test))
    print(precision_score(y_test,y_pred))
```

```
0.8733075435203095
[[790  25]
 [106 113]]
0.5159817351598174
```

```
[ ] mnb.fit(X_train,y_train)
    y_pred1=mnb.predict(X_test)
    print(accuracy_score(y_test,y_pred1))
    print(confusion_matrix(y_pred1,y_test))
    print(precision_score(y_test,y_pred1))
```

```
0.9709864603481625
[[896  30]
 [  0 108]]
1.0
```

```
[ ] bnb.fit(X_train,y_train)
    y_pred2=bnb.predict(X_test)
    print(accuracy_score(y_test,y_pred2))
    print(confusion_matrix(y_pred2,y_test))
    print(precision_score(y_test,y_pred2))
```

```
0.9835589941972921
[[895  16]
 [  1 122]]
0.991869918699187
```

FINAL OUTPUT:

```
Enter message to check: Hi there! You've been selected to receive a FREE GIFT as a loyal customer. Click the link below to claim your prize before it expires!
The message is classified as spam
```

5. NLP Implementation:

I have chosen the pre-trained model called BERT for this task.

BERT (Bidirectional Encoder Representations from Transformers)

BERT is a **deep learning model** developed by Google, based on the **Transformer architecture**. It's widely used for a variety of Natural Language Processing (NLP) tasks such as text classification, question answering, and language translation. Unlike traditional models, BERT is designed to understand the context of a word by looking at the words on both sides of it (bidirectional attention).

How BERT Works:

- **Bidirectional Attention**: Unlike earlier models that read text in one direction (left-to-right or right-to-left), BERT reads text in both directions simultaneously. This allows it to better understand the context of words within a sentence. For example, the word "bank" would be interpreted differently in "bank of a river" vs. "bank account."
- **Pre-training and Fine-tuning**:
  - **Pre-training**: BERT is pre-trained on large amounts of text, learning general language patterns by performing two tasks: **Masked Language Modeling (MLM)** and **Next Sentence Prediction (NSP)**. In MLM, BERT learns to predict masked words in sentences, and in NSP, it learns the relationship between consecutive sentences.
  - **Fine-tuning**: After pre-training, BERT is fine-tuned on specific tasks (e.g., sentiment analysis, named entity recognition) using task-specific labeled datasets.

Strengths of BERT:

- **Contextualized Understanding**: BERT's bidirectional attention allows it to grasp the context of words better than models that process text in only one direction.
- **Versatility**: It can be applied to a wide range of NLP tasks with minimal adjustments, such as classification, translation, and text generation.
- **Pre-trained Models**: BERT comes pre-trained on massive datasets, which means you can achieve high performance on your task with limited data by simply fine-tuning the model.

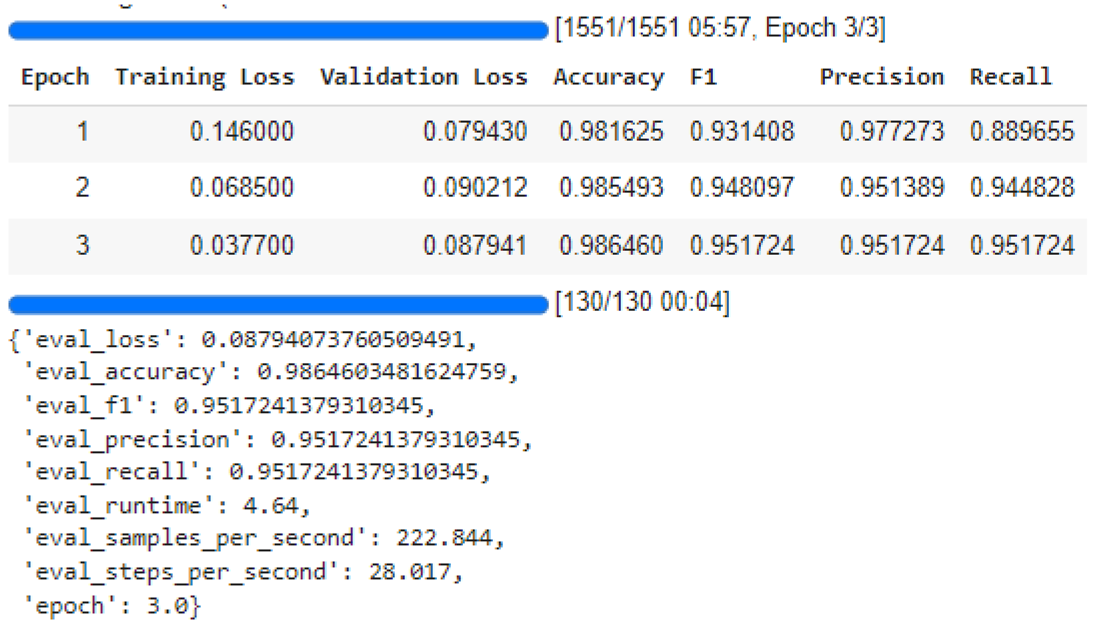
Limitations of BERT:

- **Computationally Expensive**: BERT is large and complex, requiring significant computational resources (memory, processing power) to train and even to fine-tune.
- **Slow Inference**: Due to its large size and the complexity of the Transformer architecture, BERT can be slower during inference (prediction phase), especially on devices with limited resources like mobile phones.
- **Not Always Interpretable**: Deep learning models like BERT are often referred to as "black boxes" because understanding how they arrive at specific predictions can be difficult.

Use Cases:

- **Text classification**: BERT is used for tasks like sentiment analysis (positive/negative reviews), spam detection, and more.
- **Question answering**: BERT powers systems that can answer questions based on a passage of text, like virtual assistants.
- **Named entity recognition (NER)**: Identifying proper nouns like names of people, places, and organizations in a text.

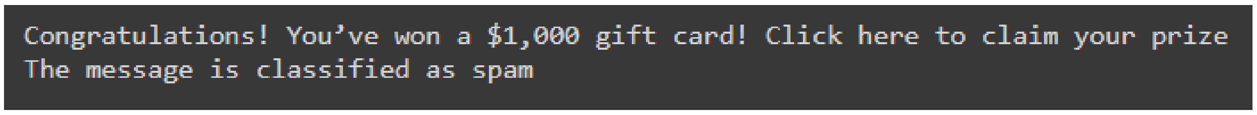
The model has been originally pre-trained using a large text corpus (Wikipedia, Britannica Encyclopedia, etc.).
I have fine-tuned the model to perform better on the spam messages dataset.
I used the T4 GPU available on Colab for computational efficiency.
The results are attached below:



[1551/1551 05:57, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.146000	0.079430	0.981625	0.931408	0.977273	0.889655
2	0.068500	0.090212	0.985493	0.948097	0.951389	0.944828
3	0.037700	0.087941	0.986460	0.951724	0.951724	0.951724

```
[130/130 00:04]
{'eval_loss': 0.08794073760509491,
 'eval_accuracy': 0.9864603481624759,
 'eval_f1': 0.9517241379310345,
 'eval_precision': 0.9517241379310345,
 'eval_recall': 0.9517241379310345,
 'eval_runtime': 4.64,
 'eval_samples_per_second': 222.844,
 'eval_steps_per_second': 28.017,
 'epoch': 3.0}
```



```
Congratulations! You’ve won a $1,000 gift card! Click here to claim your prize
The message is classified as spam
```

Difference between the working of Naive Bayes and NLP-based classification:

Naive Bayes Approach:

- **Statistical Approach**: Naive Bayes is a probabilistic classifier based on Bayes' theorem. It assumes feature independence (often called the "naive" assumption), which works well in many text classification tasks.
- **Feature Simplicity**: Naive Bayes relies on features like word frequency or the presence of certain terms. Each word's occurrence (or absence) is considered independently when making predictions.
- **Efficiency and Speed**: It's computationally lightweight and fast, suitable for real-time predictions, especially when dealing with straightforward text-based features like spam words.
- **Performance on Small Datasets**: Naive Bayes can perform reasonably well on small datasets due to its simplicity and reliance on word probabilities.

NLP (BERT) based approach:

- **Deep Learning and Contextual Understanding**: NLP-based classifiers (like BERT or other language models) use deep learning to capture the context, meaning, and syntactic structure of sentences, beyond just word frequency.
- **Pretrained Models and Fine-tuning**: NLP models often start by pretraining on large corpora, which helps them understand language context. Fine-tuning these models on spam data allows them to recognize more complex spam patterns.
- **Feature Complexity**: NLP-based classifiers represent text using token embeddings, attention mechanisms, and language semantics, allowing them to handle nuanced spam features such as sentence structure, tone, or hidden intent.

Results and Discussions:

Model	Accuracy	Precision
Bernoulli Naive Bayes(TFIDF)	0.9835	0.9918
Gaussian Naive Bayes(TFIDF)	0.8733	0.5159
Multinomial Naive Bayes(TFIDF)	0.9709	1.0
BERT-based NLP model	0.9854	0.9452

The experimental results suggest that the BERT-based NLP model shows higher accuracy in predicting the type of message.
The BERT-based NLP model is better at understanding the context behind the words in the message because it generates BERT embeddings that conceptualize the characteristics and the properties of the words in the messages which is used for fine-tuning and prediction.
However, the perfect precision attained by Multinomial Naive Bayes was not seen in the case of the NLP model.

Possible Reasons for this anomaly:

- Handling of Word Frequencies: Multinomial Naive Bayes specifically leverages word frequency, making it highly sensitive to statistically more common keywords in spam messages. This specificity helps it avoid false positives effectively, thereby obtaining improved precision. Naive Bayes can classify "spammy" terms with relative ease if they are commonly used and have been found through training.
- Contextual Nuance in NLP: NLP models, including those based on deep learning, examine contextual information beyond isolated keywords. In some cases, misclassification may result from this deeper analysis if a message has a structure similar to spam but doesn't contain the specific terms that Naive Bayes generally flags. Precision may be decreased if non-spam messages with peculiar wording are interpreted by NLP models as spam.
- Overfitting to Frequent Spam Patterns: Due to its simplicity, Multinomial Naive Bayes may overfit to frequently occurring spam patterns, producing high precision on these patterns alone. Since the NLP model may generalize more across a variety of contexts, its broader interpretation may somewhat reduce precision.

Nevertheless, the BERT model has an edge over the traditional Naive Bayes model while classifying spam messages due to its sophistication in understanding contexts and interpretation.

Learning Outcome:

- Skills Learnt:
  - Working of Naive Bayes Classification and BERT-based NLP model
  - Data preprocessing methods(Tokenization, Stemming, Lemmatization, Stop words removal)
  - Vectorizing techniques (TFIDF and Count Vectorizer)
  - Exploratory Data Analysis
- Tools used:

- Google Colab - Coding Environment
- GitHub - Track project outcomes (Code to GitHub repo)
- ChatGPT - Code explanation and Error analysis
- BERT Documentation (BERT)
- Kaggle Notebooks - SMS Spam Collection Dataset | Kaggle

Conclusion

The study investigated the efficacy of contemporary NLP methods like BERT and conventional Naive Bayes models for spam message classification.

Although the multinomial Naive Bayes model of the Naive Bayes showed great precision, the BERT model performed better overall because it was able to better understand the contextual nuances in the data, which enhanced accuracy.

Despite small differences in precision scores, the BERT model's capacity to understand semantic linkages contributed to its better overall accuracy, making it more appropriate for challenging text categorization tasks like spam detection.

Advantages:

- BERT's context-aware nature allows for more nuanced classification.
- High overall accuracy across various message types.

Limitations:

- The BERT model requires more computational resources.
- Slight discrepancies in precision compared to simpler models like Naive Bayes.

Overall, the study shows that while more basic models can provide high precision, more complex models, such as BERT, are more resilient to jobs requiring a deeper awareness of context, which makes them a better option for changing text categorization problems.