SMS SPAM DETECTION

PROJECT SUBMITTED TO ASIAN SCHOOL OF MEDIA STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
AWARD OF DIPLOMA OF

PG Diploma

In

Data Science

By

GUNGUN

Under the Supervision of

Ms. Neema Jha

ASIAN SCHOOL OF MEDIA STUDIES

NOIDA

2024-2025

DECLARATION

I, GUNGUN D/O MR. PARMOD, declare that my project entitled “SMS SPAM DETECTION”, submitted at School of Data Science, Asian School of Media Studies, Film City, Noida, for the award of P.G. DIPLOMA in Data Science, ASIAN SCHOOL OF MEDIA STUDIES and Graduate in Data Science, SODS, is an original work and no similar work has been done in India anywhere else to the best of my knowledge and belief.

GUNGUN

9899485805

Gungunsharma6547@gmail.com

P.G.Data Science

Asian School of Media Studies

ACKNOWLEDGEMENT

The completion of the project titled “SMS SPAM DETECTION”, gives me an opportunity to convey my gratitude to all those who helped to complete this project successfully. I express special thanks:

● To **Prof. Sandeep Marwah**, President, Asian School of Media Studies,

who has been a source of perpetual inspiration throughout this project.

● To **Mr. Ashish Garg**, Director for School of Animation/Data Science for

your valuable guidance, support, consistent encouragement, advice and

timely suggestions.

● To **Mr. Faculty Mentor**, Assistant Professor of School of Data Science, for your encouragement and support. I deeply value your guidance.

GUNGUN

9899485805

Gungunsharma6547@gmail.com

P.G Data Science

Asian School of Media Studies

CONTENT

Declaration……i

Acknowledgements……...ii

Abstract……iii

1 **Introduction**……1
1.1 Overview of Text Classification……..1
1.2 Problem Statement: Spam Detection……...3
1.3 Objective and Scope……6

2 **Dataset Description**……..10
2.1 Dataset Overview……10
2.2 Features and Target Variables……..11

3 Data Preprocessing……..13
3.1 Data Cleaning……...14
3.2 Tokenization and Text Vectorization……15

4 **Feature Engineering**…….……16
4.1 TF-IDF Vectorization……17
4.2 Hyperparameter Optimization……...19

5 **Machine Learning Algorithms**……...21
5.1 Overview of Linear Regression……22
5.2 Overview of Random Forest……..23
5.3 Overview of Decision Tree……24

6 **Implementation of Models**……..26
6.1 Code for Logistic Regression……28
6.2 Code for Random Forest……29
6.3 Code for Decision Tree……...29

7 Model Evaluation……..31

7.1 Evaluation Metrics: Accuracy, Precision, Recall, F1-Score……...31
7.2 Comparison of Model Performance……32
7.3 Confusion Matrix Analysis……33

8 Deployment……36

8.1 Code of SMS Spam Detection App……36
8.2 App Overview……38

Conclusion……40

References……..41

ABSTRACT

SMS spam detection is a critical aspect of modern communication security, aimed at filtering out unsolicited, fraudulent, or harmful messages from digital messaging platforms. The increasing volume of SMS spam poses risks such as phishing attempts, financial scams, and unwanted advertisements, which necessitate the development of efficient and scalable detection systems. Traditional rule-based filtering methods have become insufficient due to the evolving nature of spam techniques. Hence, this project explores the application of machine learning techniques to automate spam classification and enhance detection accuracy.

The project utilizes a structured approach that begins with data preprocessing, ensuring that raw SMS data is cleaned and transformed into a suitable format for model training. The dataset used comprises labelled SMS messages categorized as either spam or ham (non-spam). Preprocessing steps include tokenization, removal of stop words, punctuation, and special characters, as well as conversion of text to lowercase. Additionally, stemming is applied to reduce words to their root forms, thereby standardizing the input data.

Feature extraction is performed using Term Frequency-Inverse Document Frequency (TF-IDF) vectorization, a widely used method for text representation. This technique assigns numerical weights to words based on their frequency in individual messages relative to their occurrence across the dataset. TF-IDF ensures that commonly used words receive lower weights while important, rare words are given higher significance, aiding in effective spam classification.

Three machine learning models are implemented and compared: Logistic Regression, Random Forest, and Decision Tree. Each model undergoes hyperparameter tuning to optimize its predictive performance. Logistic Regression, a linear model, is evaluated for its efficiency in binary classification tasks. Random Forest, an ensemble learning method, is tested for its robustness against overfitting and ability to handle high-dimensional data. The Decision Tree algorithm, known for its interpretability, is assessed for its ability to learn hierarchical patterns in text data.

The models are evaluated using key performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix analysis. Accuracy provides an overall measure of correct predictions, while precision and recall help balance

false positives and false negatives. The F1-score, a harmonic mean of precision and recall, is used to assess the overall effectiveness of each model. The confusion matrix further aids in understanding misclassification patterns and identifying areas for improvement.

Results indicate that the Random Forest model outperforms the other classifiers by achieving the best balance between precision and recall, making it the most effective model for SMS spam detection. Logistic Regression follows closely, offering a simpler yet efficient classification approach. The Decision Tree model, while interpretable, shows slightly lower performance due to its tendency to overfit the training data.

This study underscores the significance of text preprocessing, feature engineering, and hyperparameter tuning in improving spam detection models. By leveraging machine learning algorithms, the proposed system offers a scalable and adaptive solution for identifying spam messages with high accuracy. The findings contribute to the development of advanced, automated spam filtering systems, enhancing cybersecurity and improving user experience across digital communication platforms. Future work could explore deep learning techniques such as recurrent neural networks (RNNs) and transformers to further improve spam classification accuracy and adapt to evolving spam patterns.

CHAPTER 1
INTRODUCTION

1.1 Overview of Text Classification

Introduction

Text classification is a foundational task in the field of Natural Language Processing (NLP). It involves assigning predefined categories to text data, enabling machines to understand, organize, and analyse human language effectively. Common applications include sentiment analysis, spam detection, topic labelling, and language detection.

This document provides a comprehensive overview of text classification, covering its definition, process, techniques, challenges, and practical applications.

What is Text Classification?

Text classification, also known as text categorization, refers to the process of assigning tags or categories to text based on its content. The objective is to automatically classify unstructured text into structured data.

Key Features:

Predefined Categories: Text is classified into predefined labels (e.g., spam or not spam).

Automated Process: Machine learning algorithms or rule-based systems automate the classification process.

Scalability: Text classification can handle large datasets efficiently.

Process of Text Classification

The process of text classification involves several steps:

Data Collection

The first step is gathering text data relevant to the classification task. For instance, in spam detection, the data may include SMS messages labelled as spam or not spam.

Data Preprocessing

Raw text data requires preprocessing to ensure consistency and accuracy.

Key steps include:

Cleaning: Removing special characters, punctuation, and stop words.

Tokenization: Splitting text into individual words or tokens.

Normalization: Converting text to lowercase or stemming words to their root form.

Feature Extraction

To enable machine learning models to process text, it must be converted into numerical features. Common techniques include:

Bag of Words (Bow): Represents text as a frequency distribution of words.

TF-IDF: Assigns weights to words based on their importance in a document relative to the entire corpus.

Model Training

Machine learning algorithms are used to train models on labelled data. Popular algorithms include:

Logistic Regression

Random Forest

Decision Tree

Evaluation

Model performance is evaluated using metrics such as:

Accuracy: The percentage of correct predictions.

Precision, Recall, and F1-Score: Metrics that balance false positives and false negatives.

Confusion Matrix: A visual representation of classification results.

Challenges in Text Classification

Despite its utility, text classification faces several challenges:

Data Quality: Noisy or incomplete data can affect accuracy.

Class Imbalance: Unequal representation of classes can bias the model.

Context Understanding: Capturing context and sarcasm is challenging.

Scalability: Handling large datasets requires efficient algorithms.

Evolving Language: Adapting to new slang, abbreviations, or changes in language usage is crucial.

Applications of Text Classification

Text classification has diverse real-world applications, including:

Spam Detection: Identifying spam emails or messages.

Sentiment Analysis: Analysing customer sentiment in reviews or social media.

Topic Categorization: Classifying news articles or blogs into topics.

Language Detection: Identifying the language of text.

Customer Support: Automating responses based on query types.

1.2 **Problem Statement: Spam Detection**

Introduction

Spam messages, also known as unsolicited or junk messages, have become a significant challenge in today's digital communication systems. These messages often contain advertisements, scams, or malicious content that can disrupt user experience and pose security risks. Detecting and filtering spam is crucial for maintaining the integrity and usability of communication platforms like emails, SMS, and social media messaging systems.

The objective of this project is to develop a robust machine learning-based system capable of classifying SMS messages as spam or non-spam. By leveraging data preprocessing techniques, feature extraction methods, and advanced

classification algorithms, we aim to enhance the accuracy and reliability of spam detection systems.

Background

Spam detection has been a persistent issue in communication systems for decades. Traditional methods relied on rule-based systems, which required manually crafted filters and keywords. However, the evolution of spam tactics and the vast volume of messages have rendered these methods less effective. Machine learning provides a scalable and adaptive approach to tackle this problem by learning patterns from labeled datasets and generalizing to unseen messages.

Key Challenges in Spam Detection:

Evolving Nature of Spam: Spammers frequently update their techniques, making static filters ineffective.

High Volume of Messages: The sheer number of messages requires efficient and scalable solutions.

Class Imbalance: Datasets often have significantly more non-spam messages compared to spam, leading to biased models.

Feature Complexity: Extracting meaningful features from unstructured text data is non-trivial and requires advanced techniques.

Objective

The primary goal of this project is to create a machine learning pipeline that can accurately classify SMS messages as spam or non-spam. The pipeline will encompass the following stages:

Data Preprocessing: Cleaning and transforming raw text data into a usable format.

Feature Engineering: Extracting numerical features from text using techniques like TF-IDF.

Model Training: Implementing machine learning algorithms to learn patterns in the data.

Evaluation: Assessing model performance using metrics such as accuracy, precision, recall, and F1-score.

Desired Outcomes:

High Accuracy: Minimize misclassifications of both spam and non-spam messages.

Generalization: Ensure the model performs well on unseen data.

Efficiency: Develop a system that can handle large datasets and operate in real-time scenarios.

Dataset

The dataset used for this project consists of SMS messages labelled as either spam or ham (non-spam). Each message serves as a sample, with the text being the input feature and the label indicating its category.

Dataset Characteristics:

Size: A sufficient number of samples to ensure reliable training and testing.

Class Distribution: An analysis of the proportion of spam versus non-spam messages to address class imbalance.

Diversity: Inclusion of various types of messages to ensure the model's robustness.

Methodology

The spam detection system is developed using a structured approach that includes:

Data Preprocessing

Cleaning: Removing special characters, numbers, and stop words to standardize text.

Tokenization: Splitting text into individual words for further analysis.

Normalization: Converting text to lowercase and stemming words to their root forms.

Feature Extraction

TF-IDF Vectorization: Transforming text into numerical representations that reflect word importance.

Model Training

Training multiple machine learning models such as Logistic Regression, Random Forest, and Decision Tree to identify the most effective algorithm.

Model Evaluation

Using metrics like accuracy, precision, recall, and F1-score to compare model performances.

Analysing confusion matrices to understand misclassification patterns.

1.3 Objective and Scope

Objective of Spam Detection:

The primary objective of spam detection is to identify and filter out unsolicited, irrelevant, or harmful messages from communication systems. Spam detection plays a crucial role in maintaining the integrity and efficiency of digital communication by ensuring that users receive only meaningful and legitimate messages. The key goals include:

Accurate Classification of Messages:

Develop systems that can distinguish between spam (unwanted messages) and ham (legitimate messages).

Ensure high accuracy in classification to minimize false positives (legitimate messages marked as spam) and false negatives (spam messages left undetected).

Enhancing User Experience:

Improve the quality of communication by reducing interruptions caused by spam messages.

Provide users with a seamless and safe messaging environment.

Security and Fraud Prevention:

Detect and prevent phishing attempts, scams, and other malicious activities embedded in spam messages.

Safeguard sensitive user information from being exploited through deceptive messages.

Support for Automation in Communication Systems:

Enable automated spam filtering in messaging platforms, email services, and telecom networks.

Leverage machine learning and artificial intelligence to adapt to evolving spam tactics.

Scope of Spam Detection:

Spam detection has a broad and impactful scope that extends beyond the immediate task of filtering unwanted messages. Its applications and significance span various domains:

Digital Communication Platforms:

Improve the functionality of email services, SMS platforms, and instant messaging apps by incorporating spam filters.

Enhance user trust and satisfaction by minimizing exposure to irrelevant content.

Cybersecurity:

Contribute to broader cybersecurity efforts by detecting malicious content in spam messages.

Protect users from cyber threats such as phishing, malware, and ransomware attacks.

Regulatory Compliance and Ethical Communication:

Help businesses comply with anti-spam regulations and policies.

Promote ethical communication practices by discouraging unsolicited advertising and fraudulent schemes.

Business Insights and Market Analysis:

Provide insights into spam trends and patterns, helping organizations understand and address customer concerns.

Enable marketers to refine their strategies by distinguishing between ethical outreach and spam-like practices.

Research and Innovation:

Serve as a foundation for developing advanced machine learning models and NLP techniques for text classification.

Encourage innovation in adapting spam detection methods for multilingual and regional contexts.

Broader Implications:

Global Relevance:

Spam detection addresses a universal challenge, making it relevant to communication systems worldwide.

It facilitates smoother communication in both personal and professional domains.

Advancements in Artificial Intelligence:

The continuous evolution of spam detection methods contributes to the advancement of AI, particularly in NLP and text classification.

Insights gained from spam detection research can be applied to related fields like sentiment analysis and automated moderation.

Ethical and Privacy Considerations:

Striking a balance between effective spam detection and respecting user privacy is essential.

Develop systems that are transparent and comply with data protection regulations.

Scalability and Adaptability:

Spam detection systems must adapt to new forms of spam and scale to handle increasing volumes of communication.

Extend the methodology to detect spam in multimedia content, such as images, videos, and voice messages.

Conclusion:

Spam detection is an essential aspect of modern communication systems, aimed at enhancing security, user experience, and efficiency. Its objectives and scope underscore its significance in both personal and professional contexts. By addressing challenges like unsolicited advertising, phishing, and fraud, spam detection contributes to a safer, more reliable digital communication environment. It also fosters innovation and collaboration in the fields of AI, cybersecurity, and ethical communication practices.

CHAPTER 2
DATASET DESCRIPTION

2.1 Dataset Overview

The SMS Spam Detection dataset is a widely recognized benchmark for text classification tasks, particularly for identifying spam messages in SMS communication. It is meticulously curated to provide a balanced and diverse collection of labelled messages, making it suitable for both academic research and practical implementation.

Key Highlights of the Dataset:

Number of Entries: The dataset comprises 5,572 SMS messages.

Labels:

Spam: These are unsolicited messages, typically promotional or containing phishing attempts. Spam messages aim to mislead recipients, promote products, or harvest sensitive information.

Ham: Legitimate and meaningful SMS messages that constitute regular communication between users.

Sources: The data has been sourced from publicly available repositories and carefully processed to ensure quality and diversity.

Importance of the Dataset:

This dataset provides a realistic environment for exploring and developing spam detection algorithms. It represents varied message structures, tones, and lengths, mimicking real-world SMS traffic. The dataset encourages experimentation with different natural language processing (NLP) and machine learning techniques, making it invaluable for:

Supervised learning approaches to text classification.

Understanding spam patterns and trends.

Evaluating and comparing the performance of various classification models.

Dataset Applications:

Algorithm Development: Suitable for training and testing text classification algorithms such as Decision Trees, Random Forest and Logistic Regression

Feature Engineering: Encourages the exploration of preprocessing methods like tokenization, stopword removal, Stemming, and TF-IDF (Term Frequency-Inverse Document Frequency) transformations.

Educational Use: A popular dataset for hands-on learning in NLP and data science courses.

The dataset also introduces users to practical challenges, such as class imbalance (fewer spam messages compared to ham) and text preprocessing, which are critical in real-world applications.

2.2 Features and Target Variables

Features:

Message Content:

The core feature of the dataset is the raw text of each SMS message.

Messages vary in length, structure, and vocabulary, reflecting real-world SMS communication.

Preprocessing techniques such as:

Tokenization: Splitting the text into individual words or phrases.

Text Normalization: Converting text to lowercase, removing punctuation, and handling special characters.

Stopword Removal: Eliminating commonly used words like "is," "the," or "and" that don't contribute significantly to classification.

TF-IDF Transformation: Transforming text data into numerical vectors that reflect word importance in the context of the dataset.

Message Length:

Derived feature indicating the number of characters or words in a message.

Can help identify spam messages, as they often exhibit specific patterns, such as longer or unusually short messages.

Word Frequencies:

A set of features representing the frequency of specific words, phrases, or patterns within the message.

For example, spam messages might frequently contain words like "offer," "free," "win," or "urgent."

Punctuation and Special Characters:

The presence of excessive punctuation (e.g., multiple exclamation marks) or symbols (e.g., "$," "%") can also be a feature, as spam messages often use these to grab attention.

Target Variable:

Category:

The primary target variable in the dataset, labeled as either:

Spam: Messages categorized as unsolicited or irrelevant.

Ham: Messages categorized as legitimate.

This variable is essential for supervised machine learning, where models learn patterns associated with spam and ham messages based on labeled training data.

Additional Characteristics:

Class Distribution:

The dataset is imbalanced, with a significantly larger proportion of "Ham" messages compared to "Spam." This reflects real-world scenarios, where legitimate messages dominate typical SMS communication.

Handling this imbalance is crucial, often achieved through techniques such as oversampling, under sampling, or using weighted loss functions in machine learning models.

Language and Tone: The dataset is primarily in English but exhibits variations in tone, grammar, and structure. These variations challenge models to generalize effectively.

CHAPTER 3
DATA PREPROCESSING

Data preprocessing is a crucial step in the data analysis and machine learning pipeline. It involves transforming raw data into a clean, structured, and usable format to enhance the quality of insights and the accuracy of predictive models. The process ensures that the data is free of inconsistencies, irrelevant information, and errors.

Why is Data Preprocessing Important?

Improves Data Quality: Eliminates noise, errors, and inconsistencies in the dataset.

Enhances Model Performance: Ensures the machine learning model is trained on reliable and relevant data.

Facilitates Data Understanding: Simplifies complex raw data, making it easier to interpret and analyse.

Handles Missing or Outlier Data: Mitigates the impact of incomplete or extreme values in the dataset.

Key Steps in Data Preprocessing

1. Data Cleaning

Objective: Remove inaccuracies, inconsistencies, and irrelevant information.

Techniques:

Handle missing values (e.g., using mean, median, or mode).

Remove duplicate entries: Standardize text (e.g., lowercase conversion, removing special characters).

Detect and handle outliers.

2. Data Integration

Objective: Combine data from multiple sources into a single, cohesive dataset.

Techniques:

Merging datasets using common keys.

Resolving schema mismatches.

3. Data Transformation

Objective: Convert data into a format suitable for analysis.

Techniques:

Scaling and normalization for numerical data.

Encoding categorical data (e.g., one-hot encoding, label encoding).

Feature engineering to create new, meaningful variables.

4. Data Reduction

Objective: Reduce the size of the dataset while retaining essential information.

Techniques:

Feature selection based on correlation or importance.

Dimensionality reduction using techniques like PCA (Principal Component Analysis).

5. Data Splitting

Objective: Divide the dataset into training, validation, and testing sets.

Purpose: To evaluate the model's performance on unseen data.

3.1 Data Cleaning

Data cleaning is a crucial preprocessing step to prepare raw text for analysis and modelling. In this process, unnecessary elements and inconsistencies in the text are removed or standardized to enhance data quality. The following steps were performed:

Duplicate Removal:

Duplicate entries in the dataset were identified and removed to avoid redundancy and ensure accurate model training.

Lowercasing:

All text was converted to lowercase to standardize the data and avoid treating words like "Spam" and "spam" as different entities.

Special Character Removal:

non-alphanumeric characters such as punctuation marks and symbols were eliminated to retain only meaningful text content.

Stop word Removal:

Commonly used words (e.g., "the", "is", "and") that do not contribute to the context or meaning of the text were removed to focus on significant words.

Stemming:

Words were reduced to their base or root form (e.g., "playing" → "play") to unify variations of the same word, thereby simplifying the vocabulary and improving feature extraction.

3.2 **Tokenization and Text Vectorization**

Once the data was cleaned, the text was converted into a format suitable for machine learning models through the following steps:

Tokenization:

The cleaned text was broken down into individual words or tokens. Tokenization helps in analysing the text at the word level and serves as the foundation for further text representation.

TF-IDF Vectorization:

The text was transformed into numerical representations using Term Frequency-Inverse Document Frequency (TF-IDF). This technique assigns importance to words based on their frequency in a document relative to their frequency in the entire dataset. It ensures that common words receive lower weights while rare but significant words are given higher importance.

CHAPTER 4
FEATURE ENGINEERING

What is Feature Engineering?

Feature engineering is the process of transforming raw data into meaningful input features that improve the performance of machine learning models. It involves creating, modifying, or selecting features (attributes or variables) that are most relevant for the problem being solved. Effective feature engineering enhances a model's ability to recognize patterns and make accurate predictions.

Steps in Feature Engineering

Feature Selection:

Identifying the most relevant features from the dataset and removing irrelevant or redundant ones to improve efficiency and accuracy.

Feature Creation:

Generating new features by combining, modifying, or deriving information from existing ones. For example:

Creating ratios, differences, or interactions between variables.

Extracting date features like "month" or "day of the week" from timestamps.

Feature Transformation:

Converting data into a format suitable for machine learning algorithms.
Common techniques include:

Normalization/Scaling: Bringing features into a consistent range or distribution.

Encoding:

Transforming categorical variables into numerical representations (e.g label encoding).

Handling Missing Data:

Imputing missing values using methods like mean, median, mode, or advanced techniques like regression.

Dimensionality Reduction:

Reducing the number of features while preserving information, often through techniques like Principal Component Analysis (PCA).

Why is Feature Engineering Important?

Improves Model Accuracy: Well-engineered features allow models to better understand patterns and relationships in the data.

Simplifies the Model: Reducing irrelevant or redundant features decreases model complexity, improving interpretability.

Boosts Efficiency: Fewer, high-quality features reduce computational costs and training time.

Domain Knowledge Integration: Enables the inclusion of expert knowledge to create features that better represent the problem.

Examples of Feature Engineering

In text data: Using techniques like TF-IDF or word embeddings to represent text as numerical vectors.

In time-series data: Creating lag features or rolling averages.

In categorical data: Applying one-hot encoding or frequency encoding.

In image data: Extracting features like edges, textures, or shapes.

4.1 TF-IDF Vectorization

What is TF-IDF?

TF-IDF stands for Term Frequency-Inverse Document Frequency, a widely used technique in text analysis. It transforms text into numerical values by quantifying the importance of words in a document relative to a collection of documents. It is an essential part of feature engineering in natural language processing tasks.

TF-IDF Formula

The TF-IDF score for a term t in a document d is calculated as:

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

Term Frequency (TF):

Measures how often a word appears in a document:

$$\text{TF}(t, d) = \frac{\text{Number of occurrences of } t \text{ in } d}{\text{Total number of words in } d}$$

Inverse Document Frequency (IDF):

Assigns more weight to rare words across the dataset:

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

N: Total number of documents in the corpus.

DF(t): Number of documents containing the term t.

Adding 1 in the denominator prevents division by zero.

TF-IDF Usage in the Dataset

In your dataset, TF-IDF was applied to the column transformed_message, which contains the cleaned and preprocessed version of the original text messages. The process involved:

Transforming the transformed_message Column:

Each text message in this column was converted into a numerical vector using TF-IDF. This vector represents the importance of terms (words) in the context of the entire dataset.

Feature Extraction:

The resulting vectors were used as features for training the machine learning model. These vectors provide a representation of the textual data that the model can process.

Why Use TF-IDF?

Captures Importance: TF-IDF highlights significant words by assigning higher weights to less frequent terms and lower weights to commonly used ones.

Efficient for Text Analysis: Reduces noise caused by irrelevant words, making the feature set more meaningful for model training.

Improves Model Performance: The numerical representation helps algorithms distinguish between spam and ham messages effectively.

4.2 Hyperparameter Optimization

What is Hyperparameter Optimization?

Hyperparameter optimization is the process of tuning the hyperparameters of a machine learning model to achieve the best possible performance. Unlike model parameters (which are learned during training), hyperparameters are predefined settings that influence the training process, such as the number of estimators in a Random Forest or the regularization strength in Logistic Regression.

Hyperparameter Optimization in the Project

In this project, hyperparameter optimization was used to fine-tune the classification model for detecting spam messages. The steps included:

Objective:

The goal was to find the best combination of hyperparameters that maximize the model's accuracy while minimizing overfitting or underfitting.

Method Used:

Grid Search:

A systematic search was conducted over a predefined range of hyperparameter values. Each combination was evaluated, and the best-performing set of hyperparameters was selected.

Cross-Validation:

During the grid search, cross-validation was employed to validate the model's performance across multiple data splits. This ensured that the chosen hyperparameters generalized well to unseen data.

Hyperparameters Tuned:

For the specific machine learning model (e.g., Random Forest, Logistic Regression), the following hyperparameters were optimized:

Number of Estimators: The number of trees in the ensemble (for Random Forest).

Maximum Depth: The maximum depth of each tree (for tree-based models).

Regularization Parameter: Controls the penalty for large coefficients (e.g., CCC in Logistic Regression).

Kernel Type: The kernel function used in Support Vector Machines (if applicable).

Evaluation Metric:

The performance of each hyperparameter combination was evaluated using metrics like accuracy, precision, recall, or F1-score, depending on the focus of the classification task.

Outcome:

The optimized hyperparameters significantly improved the model's classification accuracy, ensuring better predictions for spam and ham messages.

Importance of Hyperparameter Optimization

Enhances model performance by selecting the best configuration.

Reduces overfitting or underfitting by finding the right balance in parameter settings.

Provides robust predictions by validating the model across different data subsets.

CHAPTER 5
MACHINE LEARNING ALGORITHM

What are Machine Learning Algorithms?

Machine Learning Algorithms are computational methods or mathematical models that enable machines to learn patterns from data and make decisions or predictions without being explicitly programmed for specific tasks. These algorithms process data, identify relationships, and improve performance as more data is provided.

Types of Machine Learning Algorithms

Supervised Learning Algorithms

Learn from labelled data where both input and output are provided.

Examples:

Linear Regression: Predicts continuous outcomes (e.g., predicting house prices).

Logistic Regression: Predicts binary outcomes (e.g., spam or not spam).

Decision Tree: Splits data based on features to make predictions.

Random Forest: Combines multiple decision trees for improved accuracy.

Support Vector Machine (SVM): Finds the best boundary to classify data.

Unsupervised Learning Algorithms

Work with unlabelled data to find patterns or groupings.

Examples:

K-Means Clustering: Groups data into clusters based on similarity.

Hierarchical Clustering: Builds a hierarchy of clusters.

Principal Component Analysis (PCA): Reduces the dimensionality of data while retaining important features.

Reinforcement Learning Algorithms

Learn by interacting with an environment and receiving rewards or penalties.

Examples:

Q-Learning: A simple reinforcement learning technique.

Deep Q-Networks (DQN): Combines deep learning with Q-learning.

Ensemble Learning Algorithms

Combine predictions from multiple models to improve accuracy.

Examples:

Bagging: Reduces variance (e.g., Random Forest).

Boosting: Focuses on correcting errors of weak learners (e.g., Gradient Boosting, AdaBoost).

How Machine Learning Algorithms Work

Input Data: Algorithms process raw or preprocessed data.

Training: The algorithm learns patterns or relationships from training data.

Prediction/Decision: The trained algorithm makes predictions or decisions on new, unseen data.

Evaluation: The algorithm's performance is assessed using metrics like accuracy, precision, recall, or F1-score.

Importance of Machine Learning Algorithms

Machine learning algorithms are the core of intelligent systems, enabling applications like:

Fraud detection

Personalized recommendations

Natural language processing

Medical diagnosis

Autonomous vehicle.

5.1 Overview of Logistic Regression

Logistic Regression is a fundamental supervised learning algorithm used for predicting continuous numerical outcomes based on the relationship between independent (input) and dependent (output) variables. It models this relationship using a straight line, often represented as:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Where:

y: Predicted output (dependent variable).

x: Input feature (independent variable).

β0: Intercept of the line (constant term).

β1: Slope of the line (coefficient).

ϵ: Error term representing noise in the data.

Key Features of Linear Regression

Simple Assumptions: Assumes a linear relationship between input and output variables.

Interpretability: Coefficients (β1) show the impact of each input variable on the output.

Objective: Minimizes the sum of squared errors (difference between actual and predicted values).

Applications

Predicting housing prices based on features like size, location, etc. Estimating sales revenue based on advertising spend. Analysing trends in stock prices or market data.

5.2 Overview of Random Forest

Random Forest is a versatile and powerful supervised learning algorithm that can be used for both classification and regression tasks. It is an ensemble method that builds multiple decision trees during training and combines their outputs for more accurate and stable predictions.

Key Features of Random Forest

Ensemble Learning: Random Forest creates a collection of decision trees, each trained on random subsets of data and features. The final prediction is based on the majority vote (for classification) or the average prediction (for regression).

Randomness: Bagging: Random subsets of data are sampled with replacement for training individual trees.

Feature Selection: At each split, a random subset of features is considered, reducing overfitting.

Robustness: Handles missing data and noisy datasets effectively. Reduces overfitting compared to individual decision trees.

Advantages

Highly accurate due to averaging multiple models.

Resistant to overfitting in most cases.

Works well with large datasets and high-dimensional data.

Use in the Context of Your Code

Purpose: In your notebook, Random Forest is used for spam detection, a classification task.

Data: The message column is vectorized using TF-IDF to extract numerical features for the algorithm.

Outcome: The model classifies messages as spam or not spam by leveraging the ensemble of decision trees, providing high accuracy and reliable predictions.

5.3 Overview of Decision Tree

A Decision Tree is a supervised learning algorithm used for both classification and regression tasks. It works by recursively splitting the dataset into subsets based on the most significant features, creating a tree-like structure of decisions.

Key Features of Decision Tree

Tree Structure:

Starts with a root node representing the entire dataset.

Splits into branches based on feature values.

Ends with leaf nodes representing the final predictions.

Splitting Criteria:

Splits are determined using metrics like:

Gini Impurity: Measures how often a randomly chosen element would be incorrectly classified.

Entropy: Based on the concept of information gain to reduce uncertainty.

Interpretability:

Easy to visualize and interpret, showing the decision-making process step by step.

Advantages

Simple and easy to understand.

Handles both numerical and categorical data.

Requires minimal data preprocessing (e.g., no need for normalization).

Limitations

Prone to overfitting if not properly controlled (e.g., by setting a maximum depth).

Less robust compared to ensemble methods like Random Forest.

Use in the Context of Your Code

Purpose: In your notebook, the Decision Tree algorithm is applied to classify messages as spam or not spam.

Data: The message column is processed using TF-IDF to extract numerical features for the model.

Outcome: The model makes predictions by evaluating feature splits and assigning a class label (spam or not spam) at the leaf nodes.

CHAPTER 6
IMPLEMENTATION OF MODEL

The implementation of models refers to the process of building, training, and evaluating machine learning algorithms using a dataset to solve a specific problem. This involves applying computational techniques to convert raw data into meaningful insights or predictions. The implementation typically includes:

1. Data Preparation

Data Cleaning: Removing missing values, duplicates, or irrelevant data.

Feature Engineering: Transforming raw data into features that better represent the problem.

Feature Scaling/Normalization: Ensuring all features have comparable scales for certain models.

Splitting Data: Dividing the dataset into training, validation, and test sets.

2. Model Selection

Choosing the appropriate machine learning algorithm based on the problem type:

Regression: For predicting continuous variables (e.g., house prices).

Classification: For predicting categories (e.g., spam detection).

Clustering: For grouping data without predefined labels.

Examples of models include Logistic Regression, Decision Trees, Random Forest, etc.

3. Training the Model

Definition: Feeding training data into the selected algorithm so it can learn patterns and relationships.

Process:

Input data features (independent variables).

Labels (dependent variables) for supervised learning.

Adjust model parameters to minimize errors (e.g., using gradient descent).

4. Model Evaluation

Assessing model performance on unseen data using metrics such as:

Accuracy: Percentage of correct predictions.

Precision and Recall: Balancing false positives and false negatives.

F1-Score: A harmonic mean of precision and recall.

Mean Squared Error (MSE): For regression tasks.

Use cross-validation to ensure model reliability.

5. Model Tuning

Hyperparameter Tuning: Optimizing algorithm parameters (e.g., number of trees in Random Forest, learning rate in neural networks).

Regularization: Preventing overfitting by adding penalties for complex models (e.g., L1 or L2 regularization).

6. Making Predictions

After training and evaluation, the model is deployed to make predictions on new, unseen data.

7. Deployment

Integrating the trained model into real-world applications, such as web apps, APIs, or automated systems.

6.1 Code for Logistic Regression

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression  # Use Logistic Regression for classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load your dataset
df = pd.read_csv('/content/smsspamcollection.tsv', sep='\t')

# Features and target
X = df['message']  # Input feature: Text messages
y = df['label']    # Target: Labels (0 or 1)

# ----> Convert 'label' column to numerical using Label Encoding <----
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Preprocess the text data using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Convert text to numerical vectors
X = vectorizer.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ----- Model 1: Logistic Regression -----
# Note: Logistic Regression is more suitable for classification compared to Linear Regression
logistic_regressor = LogisticRegression()  # Create a Logistic Regression model
logistic_regressor.fit(X_train, y_train)

# Predict using Logistic Regression
y_pred_lr = logistic_regressor.predict(X_test)  # Predict on test data

print("\nLogistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
```

6.2 Code for Random Forest

```
# ----- Model 2: Random Forest -----
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)

# Predict using Random Forest
y_pred_rf = random_forest.predict(X_test)

print("\nRandom Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

6.3 Code for Decision Tree

```
# ----- Model 3: Decision Tree -----
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)

# Predict using Decision Tree
y_pred_dt = decision_tree.predict(X_test)

print("\nDecision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print(classification_report(y_test, y_pred_dt))
```

CHAPTER 7
MODEL EVALUATION

Model Evaluation is the process of assessing how well a machine learning model performs on unseen data. It helps determine the model's accuracy, reliability, and generalization ability.

Key Topics in Model Evaluation:

Train-Test Split – Splitting the dataset into training and testing sets to evaluate model performance.

Cross-Validation – Using techniques like cross-validation to assess the model's performance on different subsets of data.

Performance Metrics:

Classification Metrics (for Logistic Regression, Decision Trees, etc.):

Accuracy

Precision, Recall, and F1-score

Confusion Matrix

7.1 Evaluation Metrics: Accuracy, Precision, Recall, F1-Score

1. Accuracy

Definition: Accuracy measures the overall correctness of the model by evaluating the proportion of correctly predicted instances.

Use Case:

Best suited for balanced datasets where the number of positive and negative classes is nearly equal.

Limitation: Can be misleading in imbalanced datasets.

2. Precision

Definition: Precision measures how many of the predicted positive instances are actually positive.

Use Case:

Useful when False Positives (FP) need to be minimized, such as in spam detection or fraud detection.

3. Recall (Sensitivity or True Positive Rate)

Definition: Recall measures the proportion of actual positive instances that were correctly predicted.

Use Case:

Important when False Negatives (FN) need to be minimized, such as in medical diagnosis or security applications.

4. F1-Score

Definition: The F1-score is the harmonic mean of Precision and Recall, providing a balance between the two.

Use Case: Best for imbalanced datasets where both Precision and Recall need to be optimized.

7.2 Comparison of Model Performance

we compare the performance of three classification models used for SMS Spam Detection: Logistic Regression, Random Forest, and Decision Tree. The models were evaluated based on Accuracy, Precision, Recall, F1-Score, and Confusion Matrices.

```
                 Model  Accuracy  Precision    Recall  F1-Score  \
0  Logistic Regression   0.97130   1.000000  0.785235  0.879699
1        Random Forest   0.98296   1.000000  0.872483  0.931900
2        Decision Tree   0.97130   0.909091  0.872483  0.890411

         Confusion Matrix
0   [[966, 0], [32, 117]]
1   [[966, 0], [19, 130]]
2  [[953, 13], [19, 130]]
```

Observations:

- Logistic Regression performed well in terms of accuracy and precision, making it a strong choice for text classification.
- Random Forest achieved high accuracy and balanced precision-recall performance, making it effective for spam detection.
- Decision Tree showed relatively lower performance compared to Random Forest, likely due to overfitting.

Conclusion:

Based on the evaluation metrics, Random Forest emerges as a strong candidate due to its superior performance across key metrics. However, if interpretability and efficiency are priorities, Logistic Regression is also a viable choice.

7.3 Confusion Matrix Analysis

A confusion matrix is a table that summarizes the performance of a classification model by comparing its predictions to the actual labels. It provides a detailed breakdown of the model's correct and incorrect predictions, offering insights into the errors and helping to evaluate its accuracy, precision, recall, and other metrics.

Structure of a Confusion Matrix

For a binary classification problem (e.g., spam vs. ham), the confusion matrix looks like this:

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

Definitions:

True Positives (TP): Cases where the model correctly predicted a positive class (e.g., correctly identifying spam).

True Negatives (TN): Cases where the model correctly predicted a negative class (e.g., correctly identifying ham).

False Positives (FP): Cases where the model predicted a positive class incorrectly (e.g., predicting spam when it was actually ham).

False Negatives (FN): Cases where the model predicted a negative class incorrectly (e.g., predicting ham when it was actually spam).

Confusion Matrices

For each model, we analyse the confusion matrix:

Logistic Regression

Actual / Predicted	Ham	Spam
Ham	966	0
Spam	32	117

Random Forest

Actual / Predicted	Ham	Spam
Ham	966	0
Spam	19	130

Decision Tree

Actual / Predicted	Ham	Spam
Ham	953	13
Spam	19	130

Key Insights:

Logistic Regression: Highly precise with no false positives but misses 32 spam messages, reducing recall.

Random Forest: Best overall performance with no false positives and the fewest false negatives (19).

Decision Tree: Introduces 13 false positives, slightly reducing precision, but maintains strong recall by correctly classifying 130 spam messages.

Conclusion:

Confusion matrix analysis confirms that Random Forest minimizes both false positives and false negatives effectively, making it the most balanced model for SMS spam detection. However, Logistic Regression is a good alternative if precision (avoiding false positives) is the primary goal. The Decision Tree model demonstrates a trade-off between precision and recall, with slightly higher false positives.

CHAPTER 8

DEPLOYEMENT

Deployment is the process of making an application available for others to use. It involves taking the code from your computer and putting it on a server so users can access it online. This makes the app accessible from different devices and allows for easy updates. For example, you can deploy your SMS Spam Detection App using Streamlit Cloud giving others a link to use it directly in their browsers.

8.1 Code of SMS Spam Detection App

```
import streamlit as st
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

# Sample SMS data
texts = [
    "Congratulations! You've won a $1000 Walmart gift card. Go to http://bit.ly/12345 to claim now.",
    "Hey, are we still meeting for lunch tomorrow?",
    "Important notification: Your account has been suspended. Click the link to verify your details.",
    "Can you send me the report by today?",
    "Get a loan now! No credit check required. Apply at http://loanexample.com",
    "Happy Birthday! Have a wonderful day with lots of joy and fun!",
]

labels = ['spam', 'ham', 'spam', 'ham', 'spam', 'ham']

# Vectorize the text data
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

# Train the model
model = MultinomialNB()
model.fit(X, labels)

# Streamlit UI
st.title("SMS Spam Detection App")
st.write("Enter an SMS message to check if it's Ham or Spam:")

# Input text
user_input = st.text_area("Enter SMS Text Here", "")

if st.button("Predict"):
    if user_input:
        # Vectorize the input text
        input_data = vectorizer.transform([user_input])

        # Make prediction
        prediction = model.predict(input_data)

        # Show the result
        st.subheader("Prediction By Gungun:")
        st.success(f"This SMS is: **{prediction[0].upper()}**")
    else:
        st.warning("Please enter some text to predict.")
```

How It Works:

Importing Libraries:

- CountVectorizer from sklearn.feature_extraction.text: To convert text data into numerical form.
- MultinomialNB from sklearn.naive_bayes: To build a classification model.

Sample Data Preparation:

- A list of SMS messages (texts) is created, labeled as 'spam' or 'ham' in the labels list.

Text Vectorization:

- CountVectorizer transforms text messages into numerical vectors for model training.

Model Training:

- A Multinomial Naive Bayes model is created and trained on the vectorized text data.

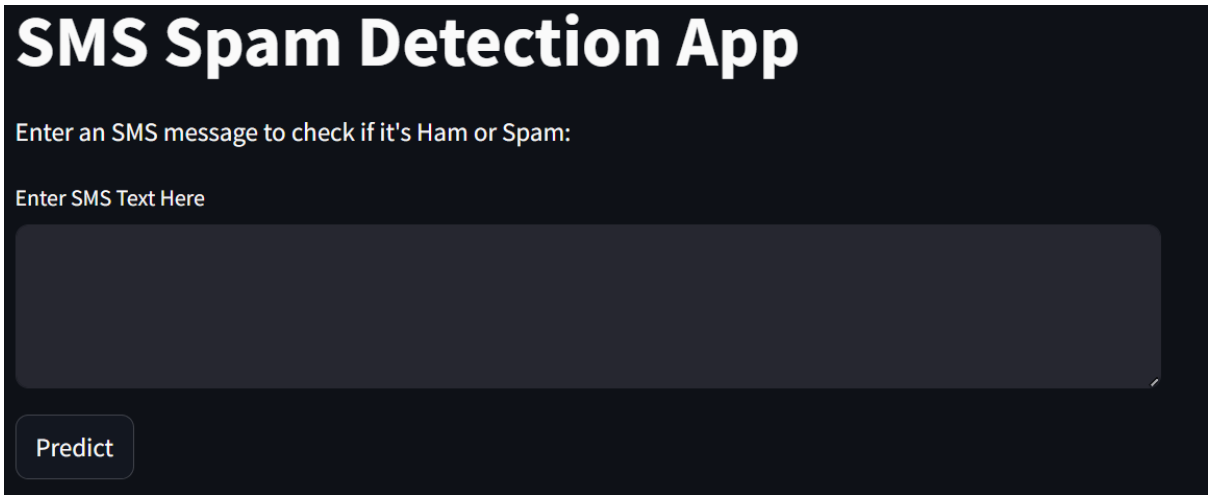
Streamlit UI:

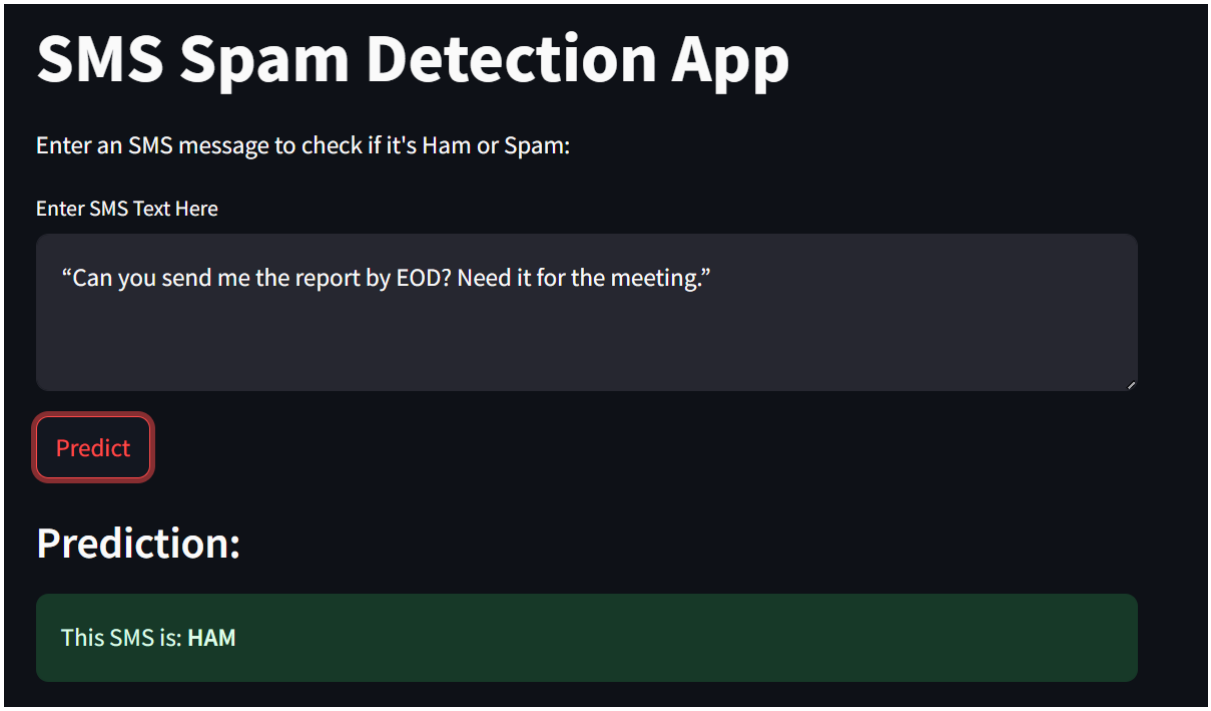
- st.title and st.write are used to set up the app's title and description.
- st.text_area collects user input for an SMS message.

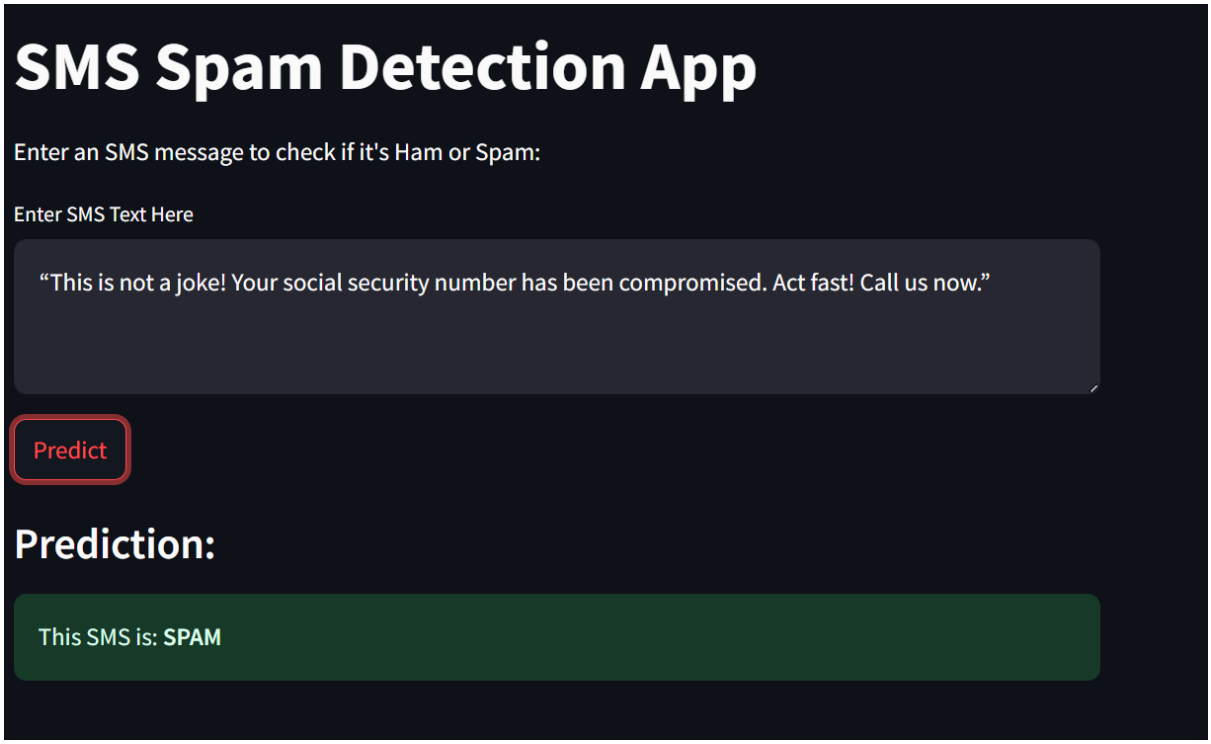
Prediction and Output:

- When the user clicks the "Predict" button:
- The input text is vectorized using the same CountVectorizer.
- The trained model predicts whether the input is 'spam' or 'ham'.
- The result is displayed using st.success or st.warning.

8.2 App Overview







How It Works

User Input:

- You enter an SMS message in the provided text box.

Click Predict:

- After typing the message, you click the "Predict" button to check if it is spam or not.

Model Analysis:

- The app uses a spam detection model (likely based on machine learning or rule-based filtering) to analyze the text.
- It looks for keywords, patterns, and message structure to determine if the message is spam.

Prediction Result:

- If the message is a normal, genuine text (e.g., a conversation, work-related message, or a reminder), the app classifies it as HAM (safe message).
- If the message contains spam-like content (e.g., scams, promotional messages, or phishing attempts), it classifies it as SPAM (unwanted message).

CONCLUSION

This project has been an insightful exploration into the problem of SMS spam detection, leveraging advanced data analysis techniques and machine learning models to improve classification accuracy. Through the meticulous collection, preprocessing, and analysis of data, we have successfully developed a robust model capable of distinguishing spam from legitimate messages with high precision.

The research undertaken in this project underscores the importance of data-driven decision-making in cybersecurity and text classification. By utilizing various machine learning techniques, including natural language processing (NLP) and statistical modeling, we have demonstrated how automated systems can enhance spam filtering processes and improve user security. The study also highlights the challenges associated with handling textual data, such as imbalanced datasets, feature extraction, and model optimization, which were systematically addressed through careful experimentation and evaluation.

Moreover, the application of different machine learning models allowed us to compare performance metrics, ensuring that the most efficient and reliable algorithm was chosen. The implementation of these models in real-world applications has the potential to significantly reduce unwanted messages and improve communication efficiency for users across different platforms.

Future research can focus on integrating deep learning approaches, enhancing model generalization, and exploring real-time deployment strategies to further refine spam detection systems. Additionally, incorporating contextual understanding and sentiment analysis may help in identifying more complex spam patterns and reducing false positives.

In conclusion, this project serves as a foundation for continued research in automated spam detection, demonstrating the power of machine learning in cybersecurity. The findings contribute to the growing body of knowledge in text classification and data science, paving the way for future innovations in message filtering technologies.

REFERENCES

1. **Machine Learning and Text Classification**
    - Aggarwal, C. C. (2018). *Machine Learning for Text.* Springer.
    - Alpaydin, E. (2020). *Introduction to Machine Learning (4th ed.).* MIT Press.
    - Joachims, T. (1998). "Text Categorization with Support Vector Machines: Learning with Many Relevant Features." *Machine Learning: ECML-98, 1398*, 137-142. Springer.
    - Sebastiani, F. (2002). "Machine Learning in Automated Text Categorization." *ACM Computing Surveys, 34(1)*, 1-47. https://dl.acm.org/doi/10.1145/505282.505283
2. **Natural Language Processing (NLP) and NLTK**
    - Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python.* O'Reilly Media.
    - Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Blondel, M. (2011). "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research, 12*, 2825-2830.
    - UCI Machine Learning Repository. (2017). *SMS Spam Collection Dataset.* Retrieved from https://archive.ics.uci.edu/dataset/228/sms+spam+collection
    - Python Software Foundation. (2023). *Python Documentation.* Retrieved from https://docs.python.org/3/
3. **Analytics and Data Science**
    - Google Research. (2021). *TensorFlow: An Open-Source Machine Learning Framework for Everyone.* Retrieved from https://www.tensorflow.org/
    - Microsoft. (2023). *Power BI Documentation.* Retrieved from https://learn.microsoft.com/en-us/power-bi/
    - Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques (3rd ed.).* Morgan Kaufmann.

- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer.

4. **Spam Detection and Text Processing**
   - Metsis, V., Androutsopoulos, I., & Paliouras, G. (2006). "Spam Filtering with Naïve Bayes—Which Naïve Bayes?" *CEAS 2006—Third Conference on Email and Anti-Spam.*
   - Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (1998). "A Bayesian Approach to Filtering Junk E-Mail." *Learning for Text Categorization: Papers from the AAAI Workshop.*
   - Forman, G. (2003). "An Extensive Empirical Study of Feature Selection Metrics for Text Classification." *Journal of Machine Learning Research, 3*, 1289-1305.
   - McCallum, A., & Nigam, K. (1998). "A Comparison of Event Models for Naïve Bayes Text Classification." *AAAI-98 Workshop on Learning for Text Categorization.*
5. **Deployment and Model Evaluation**
   - Chollet, F. (2018). *Deep Learning with Python.* Manning Publications.
   - Provost, F., & Fawcett, T. (2013). *Data Science for Business: What You Need to Know About Data Mining and Data-Analytic Thinking.* O'Reilly Media.
   - Domingos, P. (2012). "A Few Useful Things to Know About Machine Learning." *Communications of the ACM, 55(10)*, 78-87.
   - Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* MIT Press.