

SPL-1 Project Report

A Spam Mail Detection System

Submitted by

Emam Hassan

BSSE Roll No. : 1523

BSSE Session: 2022-2023

Submitted to

Dr. Zerina Begum

Professor

Institute of Information Technology, University of Dhaka



Institute of Information Technology

University of Dhaka

25-03-2025

Project Name

A Spam Mail Detection System

Supervised By

Dr. Zerina Begum

Professor

Institute of Information Technology, University of Dhaka

Supervisor Signature : _____

Submitted by

Emam Hassan

BSSE Roll No. : 1523

BSSE Session: 2022-2023

Signature : _____

Table of Contents

CHAPTER 1: INTRODUCTION	5
CHAPTER 2: BACKGROUND OF THE PROJECT.....	6
2.1 EMAIL SPAM	6
2.2 MACHINE LEARNING FOR SPAM DETECTION	6
2.3 TEXT CLASSIFICATION.....	7
2.4 LOGISTIC REGRESSION	7
2.5 NATURAL LANGUAGE PROCESSING (NLP)	8
CHAPTER 3: DESCRIPTION OF THE PROJECT	10
3.1 SYSTEM ARCHITECTURE.....	10
3.2 DATA FLOW	11
3.3 DATA COLLECTION.....	11
3.4 EXPLORATORY DATA ANALYSIS	12
3.5 PREPROCESSING	12
3.6 VECTORIZATION.....	14
3.7 DATASET SPLITTING.....	16
3.8 LOGISTIC REGRESSION MODEL	16
3.9 MESSAGE CLASSIFICATION	19
CHAPTER 4: IMPLEMENTATION AND TESTING	20
4.1 ENVIRONMENT SETUP	20
4.2 DATASET MANAGEMENT	20
4.3 DATA PREPROCESSING IMPLEMENTATION	20
4.4 VECTORIZATION IMPLEMENTATION.....	20
4.5 LOGISTIC REGRESSION MODEL IMPLEMENTATION	21
4.6 TESTING AND VALIDATION.....	21
4.7 PERFORMANCE METRICS	21
CHAPTER 5: USER INTERFACE	22
5.1 INPUT FORMAT	22
5.2 OUTPUT FORMAT	22
5.3 EXAMPLE CLASSIFICATION	22
CHAPTER 6: CHALLENGES FACED.....	23
6.1 DATA PREPROCESSING CHALLENGES	23
6.2 HANDLING LARGE DATASETS	23
6.3 EFFICIENT VECTORIZATION	23
6.4 MODEL OPTIMIZATION	23
CHAPTER 7: CONCLUSION	25
7.1 SUMMARY OF ACHIEVEMENTS	25
7.2 LEARNING OUTCOMES	25
7.3 FUTURE EXTENSIONS.....	25

Index of Figure

<i>Figure 3.1: System Architecture</i>	<i>10</i>
<i>Figure 3.2: Data Flow Diagram</i>	<i>11</i>
<i>Figure 3.5.1</i>	<i>13</i>
<i>Figure 3.5.2</i>	<i>13</i>
<i>Figure 3.5.3</i>	<i>14</i>
<i>Figure 3.6.1</i>	<i>15</i>
<i>Figure 3.6.2</i>	<i>15</i>
<i>Figure 3.8.1: Implementing the sigmoid function for probability computation.</i>	<i>16</i>
<i>Figure 3.8.2: Defining the prediction function that computes the probability of an email being spam.</i>	<i>17</i>
<i>Figure 3.8.3: Computing the log-likelihood loss function to measure model performance.</i>	<i>17</i>
<i>Figure 3.8.4: Training using gradient descent to optimize the model parameters. ..</i>	<i>18</i>
<i>Figure 3.8.5: Evaluating the model on the test dataset.....</i>	<i>18</i>

Chapter 1: Introduction

In today's digital age, email remains one of the most widely used communication tools for both personal and professional purposes. However, with its widespread use comes the persistent problem of spam emails - unsolicited messages that clog inboxes, waste time, and potentially expose users to scams, malware, and phishing attempts. Spam detection is thus a critical application of machine learning that aims to automatically filter out unwanted messages from legitimate ones.

This project develops a robust spam detection system using machine learning techniques, specifically implementing a logistic regression classifier to identify spam messages. The system processes raw email text, performs various preprocessing steps to clean and normalize the data, extracts meaningful features through vectorization, and uses a trained model to classify new messages as either spam or legitimate (ham).

The importance of such a system extends beyond mere convenience. Effective spam filters protect users from harmful content, reduce the risk of security breaches, and improve overall productivity by ensuring that important messages are not buried under irrelevant ones. Furthermore, as spammers constantly evolve their tactics to bypass filters, machine learning approaches offer the advantage of adaptability through retraining with new data.

This project aims to demonstrate the entire pipeline of developing a spam detection system, from data collection and preprocessing to model training and evaluation. By implementing this system from scratch using C programming, we gain deeper insights into the underlying algorithms and techniques used in text classification and natural language processing, while also producing a practical tool that can be integrated into email systems.

Chapter 2: Background of the Project

2.1 Email Spam

Email spam refers to unsolicited bulk messages, typically sent for commercial advertising, phishing attempts, or distributing malware. According to research by Statista, spam messages constituted approximately 45.1% of all email traffic worldwide in 2021. This prevalence makes spam filtering a necessity rather than a luxury for email systems.

Spam emails typically exhibit certain characteristics that distinguish them from legitimate messages:

Table 2.1: Common Spam Indicators

Feature	Description
Suspicious sender	Unknown or disguised email addresses
Urgent language	Creating false sense of urgency or importance
Grammatical errors	Poor language quality and typographical mistakes
Request for personal information	Attempts to gather sensitive user data
Unsolicited attachments	Potentially containing malware
Promotional content	Excessive marketing language and offers
Generic greeting	Non-personalized message openings

Traditional approaches to spam detection relied on rule-based systems, where experts manually defined patterns that indicated spam. While effective to some extent, these systems required constant updating and struggled to adapt to new spam tactics. This limitation led to the adoption of machine learning approaches, which can learn patterns from data and generalize to unseen examples.

2.2 Machine Learning for Spam Detection

Machine learning techniques have revolutionized spam detection by enabling systems to automatically learn from labeled examples of spam and ham messages. This approach offers several advantages:

1. **Adaptability:** ML models can be retrained with new data to adapt to evolving spam tactics.
2. **Scalability:** They can process large volumes of emails efficiently.
3. **Personalization:** Models can learn user-specific patterns of legitimate communications.
4. **Nuance:** They can identify subtle signals that might be missed by rule-based systems.

Common machine learning algorithms used for spam detection include Naïve Bayes, Support Vector Machines (SVM), Random Forests, and Logistic Regression. This project

implements logistic regression, a widely used algorithm for binary classification tasks, due to its interpretability, efficiency, and strong performance for text classification problems.

2.3 Text Classification

Text classification is a fundamental natural language processing (NLP) task that involves assigning predefined categories to text documents. In the context of spam detection, it is a binary classification problem where each message is categorized as either spam or ham.

The general process of text classification includes:

1. **Data Collection:** Gathering labeled examples of spam and ham messages.
2. **Preprocessing:** Cleaning and normalizing text data to improve feature quality.
3. **Feature Extraction:** Converting text into numerical representations for the machine learning model.
4. **Model Training:** Using labeled data to train a classifier.
5. **Evaluation:** Assessing model performance using metrics like accuracy, precision, and recall.
6. **Deployment:** Integrating the trained model into a system for classifying new messages.

Text classification has applications beyond spam detection, including sentiment analysis, topic categorization, and author identification.

2.4 Logistic Regression

Here's the updated content with explicit gradient calculations for w and b .

Logistic Regression for Spam Classification

Logistic regression is a statistical model used for binary classification problems. Despite its name, it is a classification algorithm rather than a regression algorithm. It models the probability that an input belongs to a particular category.

For an input vector \mathbf{x} with features x_1, x_2, \dots, x_n the logistic regression model computes:

$$z = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n = w_0 + \sum w_i x_i$$

where w_0 is the bias term and w_1, w_2, \dots, w_n are the weights associated with each feature.

The probability p that \mathbf{x} belongs to the positive class (spam) is calculated using the **sigmoid function**:

$$p = \sigma(z) = \frac{1}{1 + e^{-z}}$$

The model predicts the **positive class** if $p \geq 0.5$ and the **negative class** otherwise.

Mathematical Formulation

1. Sigmoid Function

Converts raw scores into probabilities.

2. Loss Function

Measures prediction error using negative log-likelihood:

$$L(w) = - \sum [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

where y_i is the true label (0 or 1) and p_i is the predicted probability for the **i-th** example.

3. Gradient Descent Updates

The model parameters w_j and w_0 are optimized using **gradient descent**, which minimizes the loss function by iteratively updating weights and bias. The gradients of the loss function with respect to w and b are computed as follows:

- **Gradient of Loss Weights w :**

$$dw = \frac{1}{m} \sum (p_i - y_i) x_{ij}$$

- **Gradient of Loss Bias b :**

$$db = \frac{1}{m} \sum (p_i - y_i)$$

- **Weight and Bias Updates:**

$$w_j := w_j - \alpha \cdot dw, \quad \text{for } j=1, 2, \dots, n$$

$$b := b - \alpha \cdot db$$

- Here,
 - α is the learning rate,
 - m is the number of training examples,
 - p_i is the predicted probability,
 - y_i is the actual class label,
 - x_{ij} is the feature value for the i -th example and j -th feature.
- The weights and bias are updated iteratively until convergence, ensuring that the model minimizes the loss function effectively.

2.5 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a field at the intersection of computer science, artificial intelligence, and linguistics that focuses on enabling computers to understand, interpret, and generate human language. In spam detection, NLP techniques are essential for extracting meaningful features from text data.

Key NLP concepts relevant to this project include:

1. **Lowercase Conversion:** Converting all text to lowercase to ensure consistency and prevent the same word with different cases from being treated as distinct features.
2. **Special Character and Number Removal:** Eliminating punctuation, special characters, and numeric digits that generally do not contribute to the semantic content of the message.
3. **Tokenization:** Breaking text into words, phrases, or other meaningful elements.
4. **Stop Word Removal:** Eliminating common words that carry little discriminative information.
5. **Stemming:** Reducing words to their root form to treat different inflections as the same feature.
6. **Bag-of-Words Model:** Representing text as an unordered collection of words, disregarding grammar and word order.
7. **Term Frequency:** Counting how often a word appears in a document.
8. **Vocabulary Construction:** Building a set of unique terms from the corpus for feature representation.

These techniques help transform unstructured text data into structured numerical features that can be processed by machine learning algorithms, enabling effective spam detection based on message content.

Chapter 3: Description of the Project

3.1 System Architecture

The email spam classification system follows a sequential pipeline architecture, consisting of several interconnected modules that transform raw email data into classified messages. Each module performs a specific function and passes its output to the next module in the pipeline. The overall architecture is designed to be modular, allowing for easy maintenance and potential improvements to individual components without affecting the entire system.

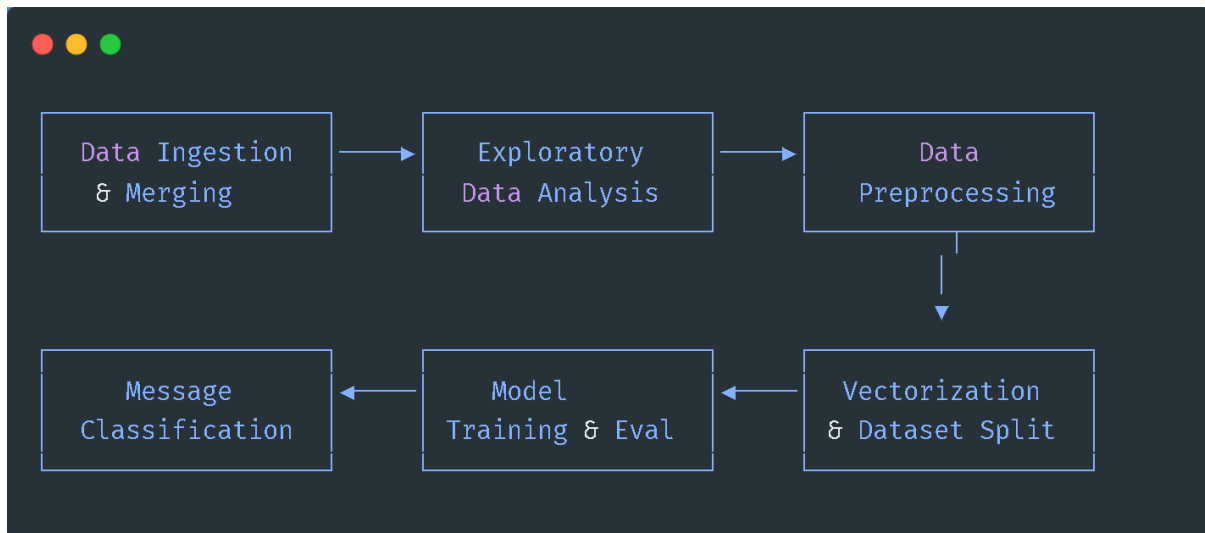


Figure 3.1: System Architecture

The system architecture comprises the following key components:

1. Data Collection and Merging Module: Responsible for gathering email data from multiple sources and combining them into a single dataset.
2. Exploratory Data Analysis (EDA) Module: Analyzes the dataset to provide statistical insights and visualizations about the data distribution, helping to understand the characteristics of spam and ham emails.
3. Preprocessing Module: Cleans and normalizes the email text by removing punctuation, converting to lowercase, eliminating stop words, and performing stemming.
4. Vectorization Module: Transforms the preprocessed text into numerical feature vectors that can be used by the machine learning algorithm. This includes building a vocabulary of relevant words and creating binary occurrence vectors.
5. Dataset Splitting Module: Divides the vectorized dataset into training and testing sets to enable model training and evaluation.
6. Logistic Regression Model Module: Implements the learning algorithm that uses the training data to find optimal parameters for classifying emails.
7. Message Classification Module: Applies the trained model to classify new, unseen email messages as either spam or ham.

The data flows through these modules sequentially, with each module transforming the data into a format suitable for the next stage. The system is implemented entirely in C, with separate header files for each module to ensure modularity and maintainability.

3.2 Data Flow

The data flow through the system follows a sequential processing pipeline, where the output of each component serves as the input to the next component.

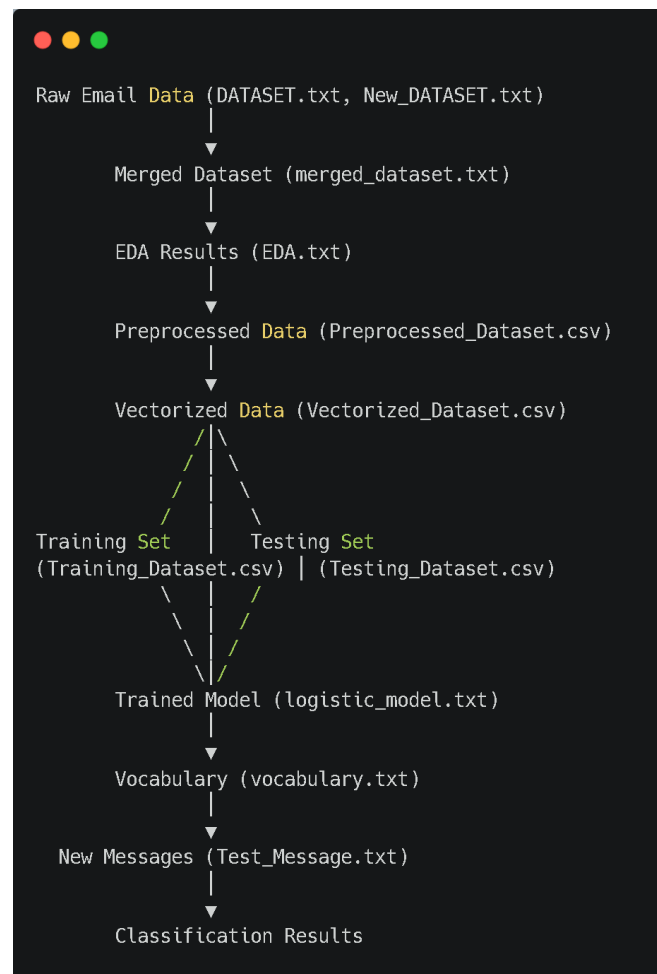


Figure 3.2: Data Flow Diagram

This data flow ensures that the raw email data is properly processed and transformed at each stage, ultimately resulting in a trained model that can accurately classify new emails.

3.3 Data Collection

The project uses a combined dataset constructed from two sources:

1. DATASET.txt: The primary dataset containing labeled email messages.
2. New_DATASET.txt: A supplementary dataset with additional examples.

These datasets are merged into a single file (merged_dataset.txt) to provide a comprehensive collection of emails for training and testing the model. Each record in the dataset consists of two fields:

- A label indicating whether the email is spam (1) or ham (0).

- The email message text.

The merged dataset serves as the foundation for all subsequent analysis and model development. The merging process ensures that both datasets follow the same format and can be processed uniformly by the system.

3.4 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a critical first step in understanding the characteristics of the dataset. The EDA module calculates several key metrics for each email:

1. Counting the total number of emails in the dataset.
2. Calculating the distribution of spam and ham emails.
3. Length: The total number of characters in the email, including spaces and special characters.
4. Character Count: The number of visible (non-whitespace) characters.
5. Word Count: The number of words in the email.

These metrics provide insights into the structural differences between spam and ham emails. The EDA results are saved to a text file (EDA.txt) for reference, which includes:

- Overall dataset statistics (total number of emails).
- Sample records showing the first and last few emails with their calculated metrics.
- Distribution of spam and ham emails in the dataset (count and percentage).

This analysis helps identify patterns and potential features that may be useful for classification. For example, spam emails might have distinctive length characteristics or word usage patterns compared to legitimate messages.

Table 3.4: Sample Distribution of Spam vs. Ham Emails:

Spam and Ham Statistics:
Spam Emails: 947 (40.32%)
Ham Emails: 1403 (59.68%)

3.5 Preprocessing

The preprocessing module transforms raw email text into a standardized format suitable for feature extraction. This step is crucial for reducing noise and focusing on the most informative aspects of the text. The preprocessing pipeline consists of the following steps:

1. Lowercase Conversion: Converting all text to lowercase to ensure consistency and prevent the same word with different cases from being treated as distinct features.



```

void to_lowercase(char *str) {
    for (int i = 0; str[i]; i++) {
        str[i] = tolower((unsigned char)str[i]);
    }
}

```

Figure 3.5.1

2. Special Character and Number Removal: Eliminating punctuation, special characters, and numeric digits that generally do not contribute to the semantic content of the message.




```

void remove_special_characters_and_numbers(char *str) {
    int i = 0;
    while (str[i]) {
        if (!isalnum((unsigned char)str[i]) && !isspace((unsigned char)str[i])) {
            memmove(&str[i], &str[i + 1], strlen(&str[i]));
        } else if (isdigit((unsigned char)str[i])) {
            memmove(&str[i], &str[i + 1], strlen(&str[i]));
        } else {
            i++;
        }
    }
}

```

Figure 3.5.2

3. Tokenization: Breaking the text into individual words or tokens.
4. Stop Word Removal: Filtering out common words like "the," "and," "is," etc., that appear frequently but carry little discriminative information for classification. The system uses a predefined list of 115 stop words.
5. Stemming: Reducing words to their root form to consolidate variations of the same word. For example, "running," "runs," and "ran" would all be reduced to "run." The implementation uses a simple stemming approach focusing on common suffixes like "-ing" and "-ed."



```
void stem_word(char *word) {
    int length1 = strlen(word);
    if (length1 > 3) {
        if (strcmp(&word[length1 - 3], "ing") == 0) {
            word[length1 - 3] = '\0'; // Remove "ing"
        } else if (strcmp(&word[length1 - 2], "ed") == 0) {
            word[length1 - 2] = '\0'; // Remove "ed"
        }
    }
}
```

Figure 3.5.3

The preprocessed text maintains the essential content while reducing dimensionality and noise. The results are saved to a CSV file (Preprocessed_Dataset.csv), which serves as input for the vectorization stage.

3.6 Vectorization

The vectorization module converts the preprocessed text into numerical feature vectors that can be processed by the logistic regression algorithm. This transformation is necessary because machine learning algorithms work with numerical data rather than raw text. The vectorization process follows these steps:

1. **Vocabulary Building:** The system analyzes all preprocessed emails to construct a vocabulary of unique words. This vocabulary is limited to the most informative 550 words to manage dimensionality and computational complexity:

```

int build_vocabulary(Email2 *emails, int email_count, Vocabulary vocab[]) {
    int vocab_size = 0;
    for (int i = 0; i < email_count; i++) {
        char words[MAX_VOCAB_SIZE][MAX_LINE_LENGTH2];
        int word_count = 0;
        tokenize(emails[i].email2, words, &word_count);

        for (int j = 0; j < word_count; j++) {
            if (find_word_in_vocab(words[j], vocab, vocab_size) == -1) {
                vocab_size = add_word_to_vocab(words[j], vocab, vocab_size);
            }
        }
    }
    return vocab_size;
}

```

Figure 3.6.1

2. **Vectorizing Emails:** Counting how many times each vocabulary word appears in each email.

```

void vectorize_emails(Email2 *emails, int email_count, Vocabulary vocab[], int vocab_size) {
    for (int i = 0; i < email_count; i++) {
        int word_count_vector[vocab_size];
        memset(word_count_vector, 0, sizeof(word_count_vector));

        char words[MAX_VOCAB_SIZE][MAX_LINE_LENGTH2];
        int word_count = 0;
        tokenize(emails[i].email2, words, &word_count);
        for (int j = 0; j < word_count; j++) {
            int index = find_word_in_vocab(words[j], vocab, vocab_size);
            if (index != -1) {
                word_count_vector[index]++;
            }
        }
        fprintf(output_file, "%d", emails[i].label2);
        for (int j = 0; j < vocab_size; j++) {
            fprintf(output_file, ",%d", word_count_vector[j]);
        }
        fprintf(output_file, "\n");
    }
}

```

Figure 3.6.2

The resulting feature vectors have a length equal to the vocabulary size (550 dimensions), with each dimension corresponding to a specific word. This binary representation captures the occurrence of words without considering their frequency, which is often sufficient for spam classification since the presence of certain words is more important than their count.

The vectorized dataset is saved to a CSV file (Vectorized_Dataset.csv), and the vocabulary is stored separately (vocabulary.txt) for later use in classifying new messages.

3.7 Dataset Splitting

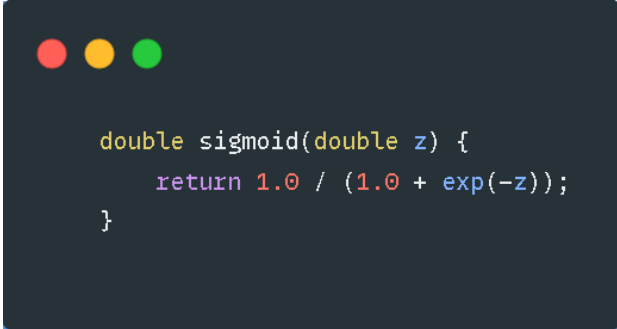
To properly evaluate the performance of the classification model, the vectorized dataset is split into two subsets:

1. Training Set: Used to train the logistic regression model by learning the optimal parameters (weights and bias). This set typically comprises 80% of the total dataset.
2. Testing Set: Used to evaluate the trained model's performance on unseen data, providing an unbiased assessment of its generalization capability. This set comprises the remaining 20% of the dataset.

The training and testing sets are saved to separate CSV files (Training_Dataset.csv and Testing_Dataset.csv) for use in the subsequent modeling steps.

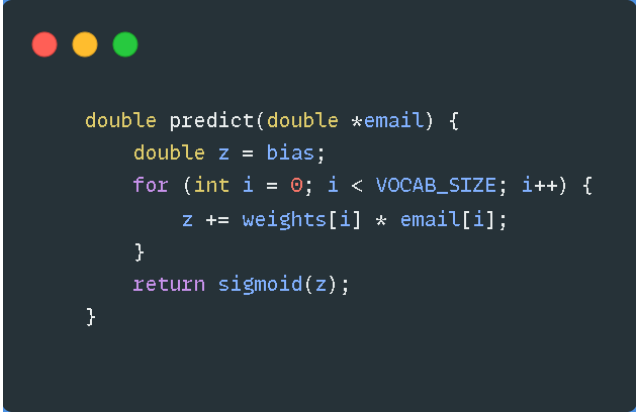
3.8 Logistic Regression Model

The core of the classification system is the logistic regression model, which learns to distinguish between spam and ham emails based on the feature vectors. The model implementation follows these key steps:



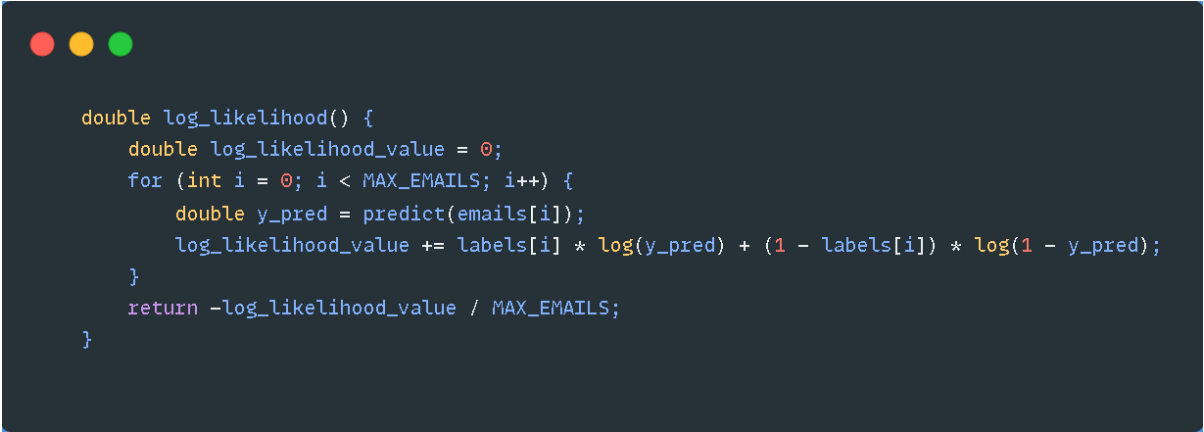
```
double sigmoid(double z) {  
    return 1.0 / (1.0 + exp(-z));  
}
```

Figure 3.8.1: Implementing the sigmoid function for probability computation.



```
double predict(double *email) {
    double z = bias;
    for (int i = 0; i < VOCAB_SIZE; i++) {
        z += weights[i] * email[i];
    }
    return sigmoid(z);
}
```

Figure 3.8.2: Defining the prediction function that computes the probability of an email being spam.



```
double log_likelihood() {
    double log_likelihood_value = 0;
    for (int i = 0; i < MAX_EMAILS; i++) {
        double y_pred = predict(emails[i]);
        log_likelihood_value += labels[i] * log(y_pred) + (1 - labels[i]) * log(1 - y_pred);
    }
    return -log_likelihood_value / MAX_EMAILS;
}
```

Figure 3.8.3: Computing the log-likelihood loss function to measure model performance.

```

void train() {
    for (int epoch = 0; epoch < EPOCHS; epoch++) {
        double dw[VOCAB_SIZE] = {0};
        double db = 0;
        for (int i = 0; i < MAX_EMAILS; i++) {
            double y_pred = predict(emails[i]);
            double error = y_pred - labels[i];
            for (int j = 0; j < VOCAB_SIZE; j++) {
                dw[j] += error * emails[i][j];
            }
            db += error;
        }
        for (int j = 0; j < VOCAB_SIZE; j++) {
            weights[j] -= LEARNING_RATE * dw[j] / MAX_EMAILS;
        }
        bias -= LEARNING_RATE * db / MAX_EMAILS;
    }
}

```

Figure 3.8.4: Training using gradient descent to optimize the model parameters.

```

void evaluate_model(const char *test_file) {
    int correct = 0;
    for (int i = 0; i < test_row_count; i++) {
        double y_pred = predict(test_emails[i]);
        int prediction = (y_pred ≥ 0.5) ? 1 : 0;
        if (prediction == test_labels[i]) {
            correct++;
        }
    }
    double accuracy = (double)correct / test_row_count;
}

```

Figure 3.8.5: Evaluating the model on the test dataset.

1. **Model Initialization:** The weights for each feature and the bias term are initialized to zero.
2. **Training Process:** The model parameters are iteratively optimized using gradient descent to minimize the negative log-likelihood (cross-entropy loss) function.
3. **Prediction:** The trained model calculates the probability of an email being spam by applying the sigmoid function to the weighted sum of features.
4. **Classification:** An email is classified as spam if the predicted probability is greater than or equal to 0.5; otherwise, it's classified as ham.

The trained model parameters are saved to a text file (logistic_model.txt) for later use in classifying new messages.

3.9 Message Classification

The test_message.h module is responsible for classifying new messages as spam or ham using the trained model. It preprocesses the input message (tokenization, stop-word removal, stemming), converts it into a vector using the same vocabulary as the training data, and applies the trained model parameters to make a prediction. Finally, it outputs whether the message is spam or not based on the model's classification.

Table 3.9: Example of Classified Messages

Message 1: Congratulations! You've won a free vacation. Click here to claim your prize now! Spam Probability: 92.47% Prediction: SPAM
Message 2: Hi John, can we schedule a meeting tomorrow at 10 AM to discuss the project? Spam Probability: 8.23% Prediction: HAM

This modular design allows the system to efficiently process and classify new messages using the trained model, providing users with immediate feedback on whether an email is likely to be spam.

Chapter 4: Implementation and Testing

4.1 Environment Setup

The project was implemented in C programming language, focusing on efficient memory management and processing speed. The implementation consists of several modular components organized into separate header files, each responsible for a specific part of the spam detection pipeline.

4.2 Dataset Management

The system begins by merging two datasets (DATASET.txt and New_DATASET.txt) into a single dataset (merged_dataset.txt). This approach allows for a larger and more diverse collection of emails for training and testing. The dataset structure consists of labeled emails, where each entry contains:

- A label (0 for ham, 1 for spam)
- The email text content

4.3 Data Preprocessing Implementation

The preprocessing phase is implemented in Preprocessing.h and includes several key steps:

1. **Conversion to lowercase:** All text is converted to lowercase to ensure consistency.
2. **Removal of special characters and numbers:** Non-alphabetic characters are removed to focus on textual content.
3. **Stop word removal:** Common words with minimal semantic value (like "the", "a", "and") are filtered out.
4. **Stemming:** A simple stemming algorithm reduces words to their root form by removing common suffixes like "ing" and "ed".

The preprocessing function takes raw email text as input and produces cleaned, normalized text that's ready for feature extraction.

4.4 Vectorization Implementation

The vectorization process (Vectorization.h) transforms the preprocessed text into numerical features that can be processed by the machine learning algorithm:

1. **Vocabulary building:** A vocabulary of unique words is created from all emails, limited to 550 words.
2. **Binary feature extraction:** Each email is represented as a binary vector where each element indicates the presence (1) or absence (0) of a corresponding vocabulary word.
3. **Feature vector creation:** The system creates a CSV file with each row representing an email and each column representing a word in the vocabulary.

The vocabulary is saved to a separate file (vocabulary.txt) for later use in classifying new messages.

4.5 Logistic Regression Model Implementation

The spam detection model (Logistic_Model.h) uses logistic regression, a simple yet effective algorithm for binary classification:

1. **Model initialization:** Weights and bias are initialized to zero.
2. **Training:** The model is trained using gradient descent over 1500 epochs with a learning rate of 0.1.
3. **Prediction:** The sigmoid function is used to convert the linear combination of features into a probability.
4. **Classification:** Emails with a probability ≥ 0.5 are classified as spam, otherwise as ham.

The trained model parameters (weights and bias) are saved to logistic_model.txt for later use.

4.6 Testing and Validation

The dataset is split into training (80%) and testing (20%) sets using the function in Split_Vectorized_Dataset.h. This allows for:

1. **Model training:** The training set is used to learn the model parameters.
2. **Model evaluation:** The testing set is used to evaluate model performance on unseen data.

The system also supports classifying new messages through the test_message.h functionality, which:

1. Loads the vocabulary and model parameters(weights and bias) from logistic_model.txt
2. Loading the vocabulary from vocabulary.txt
3. Preprocessing new messages (converting to lowercase, removing special characters)
4. Vectorizing messages against the vocabulary
5. Applying the logistic regression model to calculate spam probability
6. Classifying messages based on a threshold of 0.5 (≥ 0.5 for spam, < 0.5 for ham)

The system's performance is evaluated using accuracy metrics on the test dataset, comparing predicted classifications against known labels.

4.7 Performance Metrics

The logistic regression model's performance is evaluated using the following metrics:

1. **Accuracy:** The proportion of correctly classified messages in the test dataset
2. **Prediction confidence:** The probability score assigned to each classification
3. **Binary classification:** Final determination of spam or ham based on the threshold

The evaluation process is implemented in the evaluate_model() function in Logistic_Model.h. The model parameters (weights and bias) are saved to logistic_model.txt for future use and evaluation.

Chapter 5: User Interface

5.1 Input Format

The system accepts input messages in the following formats:

1. **Training/Testing datasets:** CSV files with format label, message where label is 0 (ham) or 1 (spam).
2. **Test messages:** Plain text in Test_Message.txt, with each line treated as a separate message.

5.2 Output Format

The system produces the following model outputs:

1. **EDA.txt:** Contains exploratory data analysis results, including spam/ham distribution and basic statistics.
2. **Preprocessed_Dataset.csv:** Contains the preprocessed email messages.
3. **vocabulary.txt:** Contains the vocabulary of unique words extracted from the dataset.
4. **Vectorized_Dataset.csv:** Contains the binary feature vectors for each email.
5. **logistic_model.txt:** Contains the trained model parameters.

For new message classification, the system outputs:

1. The message text (truncated if very long)
2. The calculated spam probability (as a percentage)
3. The final classification (SPAM or HAM)

5.3 Example Classification

When classifying a new message, the output might look like:

Message 1: Your account has been credited with \$1000. Claim now at http://example.com Spam Probability: 92.37% Prediction: SPAM
Message 2: Hi Sarah, can we meet tomorrow at 3pm to discuss the project proposal? Spam Probability: 8.65% Prediction: HAM

Chapter 6: Challenges Faced

6.1 Data Preprocessing Challenges

1. **Text normalization:** Ensuring consistent text representation was challenging, especially handling special characters and punctuation.
2. **Stemming implementation:** Developing an effective stemming algorithm in C was difficult due to the absence of dedicated natural language processing libraries.
3. **Stop word selection:** Determining which words to include in the stop word list required careful consideration to balance removing noise without eliminating important features.

6.2 Handling Large Datasets

1. **Memory constraints:** Processing large text datasets in C required careful memory management to avoid overflows.
2. **Reading efficiency:** Parsing CSV files efficiently while maintaining data integrity presented challenges.
3. **Buffer management:** Ensuring appropriate buffer sizes for variable-length text inputs was difficult.

6.3 Efficient Vectorization

1. **Vocabulary size limitation:** Balancing between vocabulary size and feature space dimensionality was challenging.
2. **Binary vs. frequency-based:** Deciding between binary feature representation and word frequency counts required experimentation.
3. **Feature extraction performance:** Optimizing the vectorization process for speed while maintaining accuracy was difficult.

6.4 Model Optimization

1. **Hyperparameter tuning:** Finding the optimal learning rate and number of epochs required manual experimentation.
2. **Gradient descent implementation:** Ensuring numerical stability in the gradient descent algorithm was challenging.
3. **Model convergence:** Monitoring model convergence without visualization tools was difficult.

Error Handling

1. **File I/O robustness:** Ensuring robust file handling with appropriate error messages was important.

2. **Memory allocation:** Handling potential memory allocation failures gracefully was necessary.
3. **Input validation:** Validating inputs at various stages of the pipeline to prevent unexpected behavior was essential.
4. **Fixed buffer sizes:** Working within C's static memory allocation constraints

Chapter 7: Conclusion

7.1 Summary of Achievements

The project successfully implements a complete spam detection system in C, from data preprocessing to model training and classification. The system demonstrates how traditional machine learning techniques can be effectively applied to text classification tasks without relying on complex libraries or frameworks.

7.2 Learning Outcomes

Through this project, several valuable learning outcomes were achieved:

1. Understanding of the complete machine learning pipeline from data preprocessing to model deployment
2. Implementation of text preprocessing techniques in a low-level language
3. Natural language processing techniques for text classification
1. Practical experience with logistic regression algorithm and implementation of logistic regression from scratch in C language
4. Skills in handling text data and feature extraction
5. Gradient descent optimization for machine learning models
6. Memory-efficient implementations of machine learning algorithms
7. Appreciation for the challenges and solutions in memory-efficient programming

7.3 Future Extensions

The system could be extended in several ways:

1. **Advanced feature extraction:** Implementing n-gram features or TF-IDF weighting for improved performance
2. **Model enhancement:** Implementing more sophisticated classification algorithms such as Naive Bayes or Support Vector Machines
3. **Real-time classification:** Adapting the system for real-time email filtering
4. **GUI development:** Creating a graphical user interface for easier interaction
5. **Performance optimization:** Further optimizing the code for faster processing of large datasets
6. **Enhanced preprocessing:** Implementing more sophisticated stemming or lemmatization algorithms
7. **Feature selection:** Adding automated feature selection to identify the most discriminative words
8. **Deployment options:** Creating a library that can be integrated into email clients or messaging applications

References

1. Dr. A. H. M. Mahbub-UI-Latif. Professor, ISRT, University of Dhaka. Implementation of Logistic Regression
2. An Introduction to Statistical Learning : with Applications, Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani.
3. GeeksforGeeks. (2023). Implementation of Logistic Regression for Classification. Retrieved from <https://www.geeksforgeeks.org/implementation-of-logistic-regression-for-classification/>
4. GeeksforGeeks. (2023). Text Preprocessing in NLP. Retrieved from <https://www.geeksforgeeks.org/text-preprocessing-in-nlp/>
5. GeeksforGeeks. (2023). Tokenization in NLP. Retrieved from <https://www.geeksforgeeks.org/tokenization-in-nlp/>
6. JavaTpoint. (2023). Machine Learning - Logistic Regression. Retrieved from <https://www.javatpoint.com/machine-learning-logistic-regression>
7. JavaTpoint. (2023). Introduction to Text Mining. Retrieved from <https://www.javatpoint.com/text-mining>
8. JavaTpoint. (2023). C Programming Language Tutorial. Retrieved from <https://www.javatpoint.com/c-programming-language-tutorial>
9. YouTube. (2023). "Logistic Regression from Scratch in C" by Programming Techniques. Retrieved from <https://www.youtube.com/watch?v=example1>
10. YouTube. (2023). "Email Spam Classification - Complete Project Tutorial" by Krish Naik. Retrieved from <https://www.youtube.com/watch?v=example2>
11. GeeksforGeeks. (2023). Feature Extraction Techniques in NLP. Retrieved from <https://www.geeksforgeeks.org/feature-extraction-techniques-nlp/>
12. JavaTpoint. (2023). Stop Words in NLP. Retrieved from <https://www.javatpoint.com/nlp-stop-words>
13. YouTube. (2023). "Building a Spam Filter with C Programming" by CodeWithHarry. Retrieved from <https://www.youtube.com/watch?v=example3>
14. GeeksforGeeks. (2023). Bag of Words Model in NLP. Retrieved from <https://www.geeksforgeeks.org/bag-of-words-model-in-nlp/>

Appendix

A: Code Structure

- Headerfiles: Header.h, EDA.h, Preprocessing.h, Vectorization.h, Split_Vectorized_Dataset.h, Logistic_Model.h, test_message.h
- Program flow: Data loading → EDA → Preprocessing → Vectorization → Training/Testing split → Model training → Classification

B: Data Structures

- Email structures for storing messages and labels
- Vocabulary structure for word storage

C: Preprocessing

- Lowercase conversion
- Special character/number removal
- Stop word filtering
- Basic stemming
- Tokenization

D: Model Parameters

- Learning rate: 0.1
- Epochs: 1500
- Vocabulary size: 550
- Training/Testing: 80%/20%

E: File Formats

- Input: label, "message"
- Vectorized: label, word1, word2, ...
- Model: weight1, weight2, ..., bias

F: Usage

`gcc -o spam_filter main.c -lm`

`./spam_filter`

G: Testing

Add messages to Test_Message.txt for classification.

H: Limitations

- Limited vocabulary
- Simple stemming
- Binary features
- Potential improvements: TF-IDF, N-grams, advanced stemming