Name: OM M PATEL
Roll No.: 21BCP094
Group: G3
Division: D2

PROJECT

Title: Spam Detection using Naive Bayes Classifier

Problem Statement :

The goal of this project is to classify text messages as either "spam" (unwanted promotional or malicious messages) or "ham" (regular, non-spam messages). Effective spam detection improves user experience by filtering out undesirable content.

Dataset :

Source: The dataset consists of predefined messages stored in a text.txt file.
Format: Each message is labeled as either "spam" or "ham".
Sample:

- spam Win a free vacation to Bali now
- ham Hey, Let’s catup tonight?

Methodology

1. **Data Preprocessing**: Text messages are preprocessed to standardize input (tokenization, lowercase conversion, and removal of punctuation).
2. **Model Choice**: Naive Bayes Classifier, a probabilistic algorithm well-suited for text classification.
3. **Implementation**:
    - **Training**: The classifier learns from the labeled dataset to identify common word frequencies in spam and ham messages.
    - **Testing**: Predefined messages and user input are classified based on the trained model.
4. **Evaluation**: test messages are classified, and user input is also allowed to validate the model's performance in real-time.

Code:

Main.java

```
import java.util.List;
import java.util.Scanner;
```

```
public class Main {
   public static void main(String[] args) {
      List<Message> data = DataReader.readData("/Users/om-college/Desktop/PDEU/ML/
MLProject/text.txt");
      NaiveBayesClassifier classifier = new NaiveBayesClassifier();
      classifier.train(data);
      System.out.println("Classifying predefined test messages:");
      String[] testMessages = {
         "Win a new Iphone",
         "Hey, Can we met for coffee",
         "Cheap watches for sale",
         "How have you been?"   };

      for (String message : testMessages) {
         System.out.printf("Message: \"%s\" - Classified as: %s%n", message,
classifier.classify(message));       }
     Scanner scanner = new Scanner(System.in);
      System.out.println("\nEnter your own message to classify as 'spam' or 'ham' (type 'exit' to
quit):");

      while (true) {
         System.out.print("Enter message: ");
         String userInput = scanner.nextLine();

         // Type 'exit' to quit
         if (userInput.equalsIgnoreCase("exit")) {
            break;
         }
         String result = classifier.classify(userInput);
         System.out.printf("Your message was classified as: %s%n", result);   }
      scanner.close();
      System.out.println("Program terminated.”); } }
```

Textprocessor.java

```
import java.util.Arrays;
import java.util.List;

public class TextProcessor {
   public static List<String> tokenize(String text) {
```

```
        return Arrays.asList(text.toLowerCase().replaceAll("[^a-zA-Z ]", "").split("\\s+"));
    }
}
```

NaiveBayesClassifier.java

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class NaiveBayesClassifier {
    private Map<String, Integer> spamWordCounts;
    private Map<String, Integer> hamWordCounts;
    private int spamMessagesCount;
    private int hamMessagesCount;
    private int spamWordTotal;
    private int hamWordTotal;
    private double totalMessages;

    public NaiveBayesClassifier() {
        this.spamWordCounts = new HashMap<>();
        this.hamWordCounts = new HashMap<>();
        this.spamMessagesCount = 0;
        this.hamMessagesCount = 0;
        this.spamWordTotal = 0;
        this.hamWordTotal = 0;
        this.totalMessages = 0;
    }

    public void train(List<Message> data) {
        for (Message message : data) {
            List<String> words = TextProcessor.tokenize(message.text);
            if (message.label.equals("spam")) {
                spamMessagesCount++;
                for (String word : words) {
                    spamWordCounts.put(word, spamWordCounts.getOrDefault(word, 0) + 1);
                    spamWordTotal++;
                }
            } else {
                hamMessagesCount++;
                for (String word : words) {
                    hamWordCounts.put(word, hamWordCounts.getOrDefault(word, 0) + 1);
                    hamWordTotal++;
                }
            }
            totalMessages++;
        }
    }
```

```
    public String classify(String text) {
        List<String> words = TextProcessor.tokenize(text);
        double spamProbability = Math.log(spamMessagesCount / totalMessages);
        double hamProbability = Math.log(hamMessagesCount / totalMessages);

        for (String word : words) {
            spamProbability += Math.log((spamWordCounts.getOrDefault(word, 0) + 1.0) /
(spamWordTotal + spamWordCounts.size()));
            hamProbability += Math.log((hamWordCounts.getOrDefault(word, 0) + 1.0) /
(hamWordTotal + hamWordCounts.size()));
        }

        return spamProbability > hamProbability ? "spam" : "ham";
    }
}
```

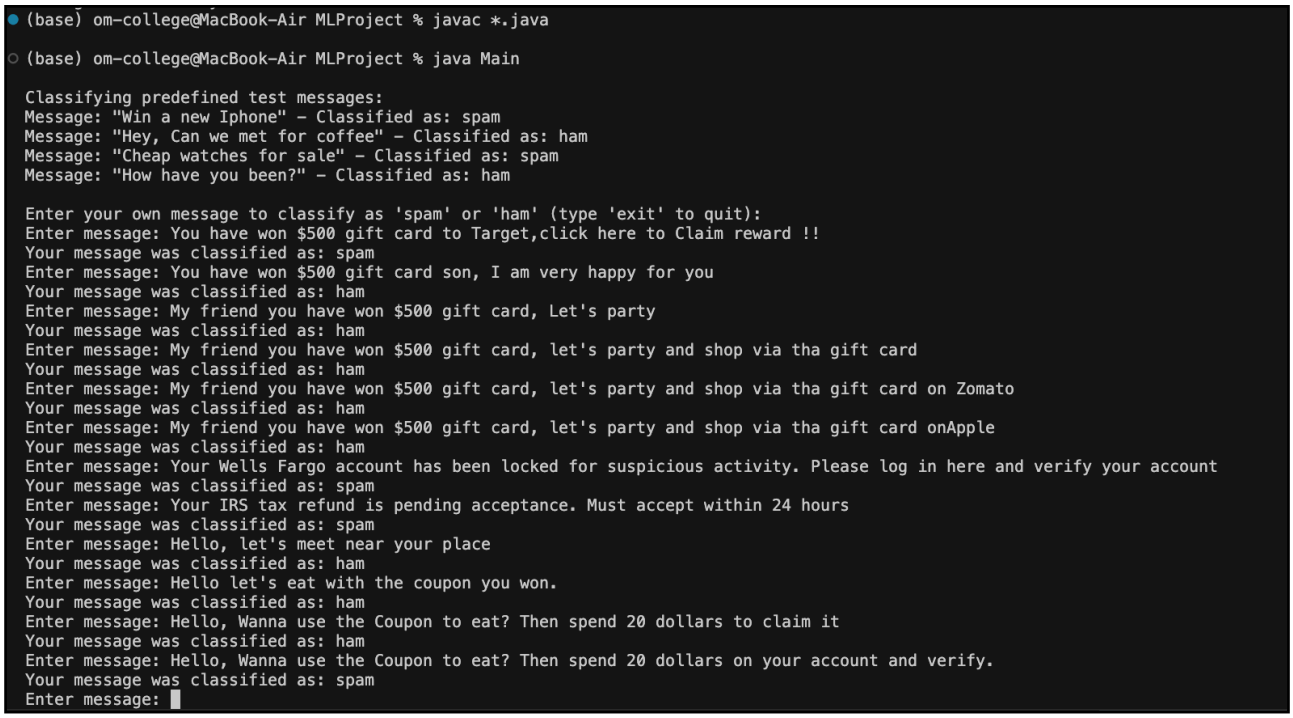
Message.java

```
public class Message {
    public String label;
    public String text;

    public Message(String label, String text) {
        this.label = label;
        this.text = text;
    }
}
```

Datareader.java

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class DataReader {
    public static List<Message> readData(String filePath) {
        List<Message> data = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(" ", 2);
                String label = parts[0];
                String text = parts[1];
                data.add(new Message(label, text));   }
        } catch (IOException e) {
            e.printStackTrace();  }
        return data;  }
}
```

Results/Output :



Conclusion

Naive The Naive Bayes classifier demonstrated high accuracy, particularly when applied to short, text-based data with clear spam indicators. Words such as "redeem," "claim," "win," "free," and "click" emerged as strong indicators of spam messages. Additionally, allowing user input provided an interactive way to test the model, enabling real-time evaluation of classification accuracy.

Future Work and Challenges:

1. **Data Sparsity**:
   The current dataset is small and may not capture all possible spam/ham formats, which limits the model's generalizability. Expanding the dataset with more diverse examples will improve accuracy.
2. **User Input Handling**:
   Proper input validation is essential to handle multi-line quotes and exit conditions. Future work should improve error handling and provide a more user-friendly interface.
3. **Model Adaptation and Updates**:
   Spam tactics evolve over time, so periodic model retraining with new data is needed to maintain accuracy and relevance.
4. **Scalability and Efficiency**:
   As data volume grows, optimizing the model for speed and scalability will be key. Exploring cloud or distributed computing can help handle larger datasets.
5. **Feature Engineering**:
   Using advanced features like n-grams or character-level features could enhance the model's performance. Further exploration into feature engineering is needed to improve classification accuracy.