معهد القاهرة العالي
للهندسة وعلوم الحاسب والإدارة
CHI

Spam Detection AI Model in Email Documentation

Name model : Spam detection

. مينا يوسف عزت	2022030112
. احمد عماد عبدالنبي	2022030128
. عاصم خالد النشار	2022030117
. ابوبكر رمضان نفادي ابوبكر	2022030187
. ملك اشرف محمد مسعد	2022030114
. منه الله رمضان احمد	2022030188

Table of Contents

Overview

Spam detection in emails refers to the process of identifying unwanted or unsolicited messages (spam). These messages often contain advertisements, phishing attempts, or malware. The goal of an AI-based spam detection system is to automatically classify these messages and filter them out, improving user experience and security.

An AI-based spam detection model typically uses machine learning algorithms, particularly those in natural language processing (NLP), to classify incoming messages as either spam or non-spam (ham). This process involves preprocessing email data, extracting features from the text, training a model, and evaluating its performance.

This document outlines how to build and deploy an AI spam detection model for email messages, with code examples in Python for each step. The core of the system is built using a dataset of SMS messages and utilizes the Naive Bayes classification algorithm to predict message categories based on input text.

1. System Architecture

The Spam Detection AI system follows a typical architecture for text classification tasks. The system is composed of three main components:

1. **Data Collection and Preprocessing**: The dataset is loaded, cleaned, and preprocessed to convert raw SMS data into a suitable format for analysis.
2. **Feature Extraction**: Text data is transformed into numerical feature vectors using a vectorization technique such as CountVectorizer.
3. **Model Training and Prediction**: A Naive Bayes model is trained on the feature vectors to classify the messages as spam or ham. The model is evaluated on a test set, and predictions can be made in real-time.

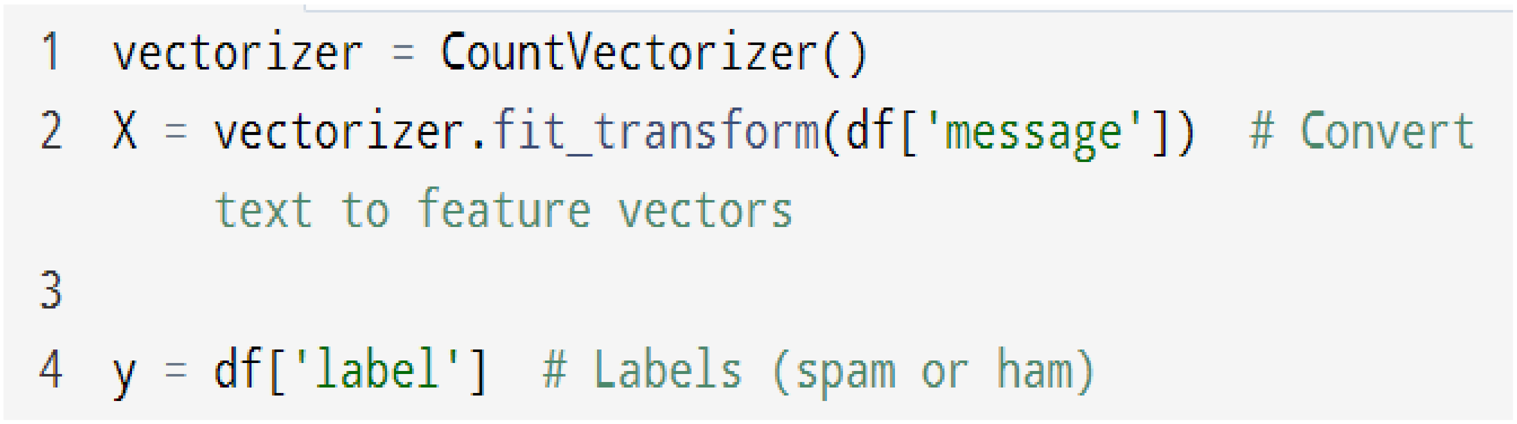
2. Data Collection and Preprocessing

The dataset used for training the model is sourced from the UCI Machine Learning Repository, specifically the SMSSpamCollection dataset. This dataset contains a collection of SMS messages labeled as either "spam" or "ham."

- The data is loaded from a zip file available at the following URL: https://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspamcollection.zip
- The dataset is then read into a pandas DataFrame, with the text messages and their corresponding labels (spam/ham).
- Labels are converted into binary values:
  - 'spam' → 1
  - 'ham' → 0

3. Feature Extraction

To make text data understandable for machine learning models, it needs to be transformed into numerical features. The **CountVectorizer** is used to convert the messages into a bag-of-words representation, which is then converted into feature vectors. Each word is treated as a feature, and the vectorizer counts the frequency of each word in the message.

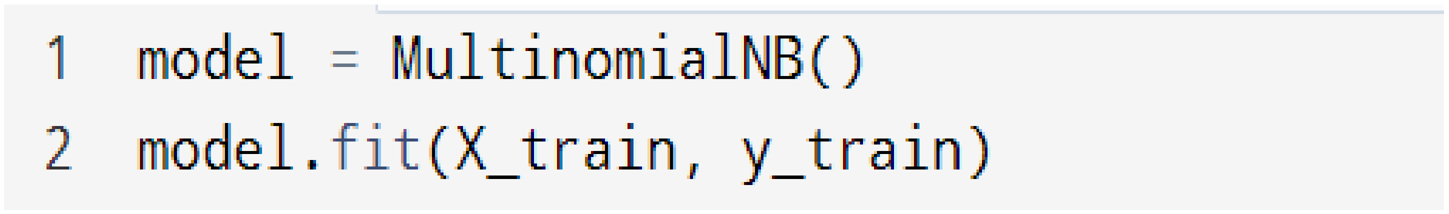


```python
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['message'])  # Convert text to feature vectors

y = df['label']  # Labels (spam or ham)
```

4.Model Training and Evaluation

The model used for classification is the Multinomial Naive Bayes (MNB) classifier, which is a popular algorithm for text classification tasks.

- **Training:** The dataset is split into training and testing sets using train_test_split. The training set is used to train the Naive Bayes model.
- **Evaluation:** After training, the model is tested on the unseen test data, and the performance is evaluated using metrics like accuracy and classification report.

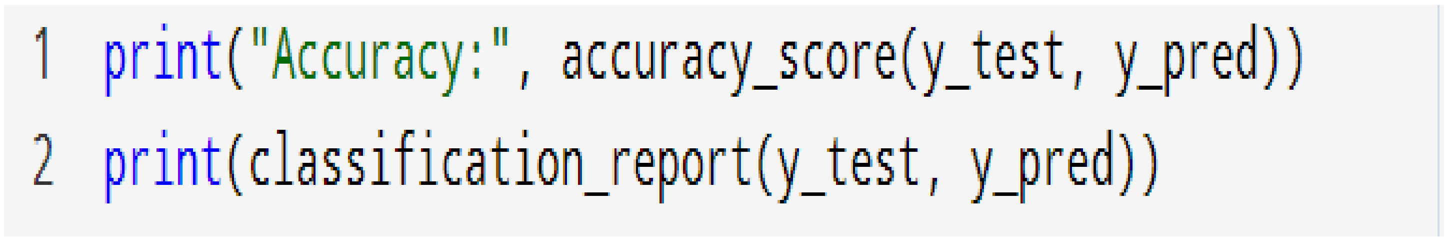


```
model = MultinomialNB()
model.fit(X_train, y_train)
```

5. Evaluation Metrics

To assess the performance of the spam detection model, we use several metrics:

- **Accuracy:** The percentage of correct predictions made by the model.
- **Precision:** The proportion of true positives (spam messages) among all predicted spam messages.
- **Recall:** The proportion of true positives (spam messages) among all actual spam messages.
- **F1-Score:** The harmonic mean of precision and recall, providing a balanced evaluation.



```python
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

6. Deployment and Real-time Filtering

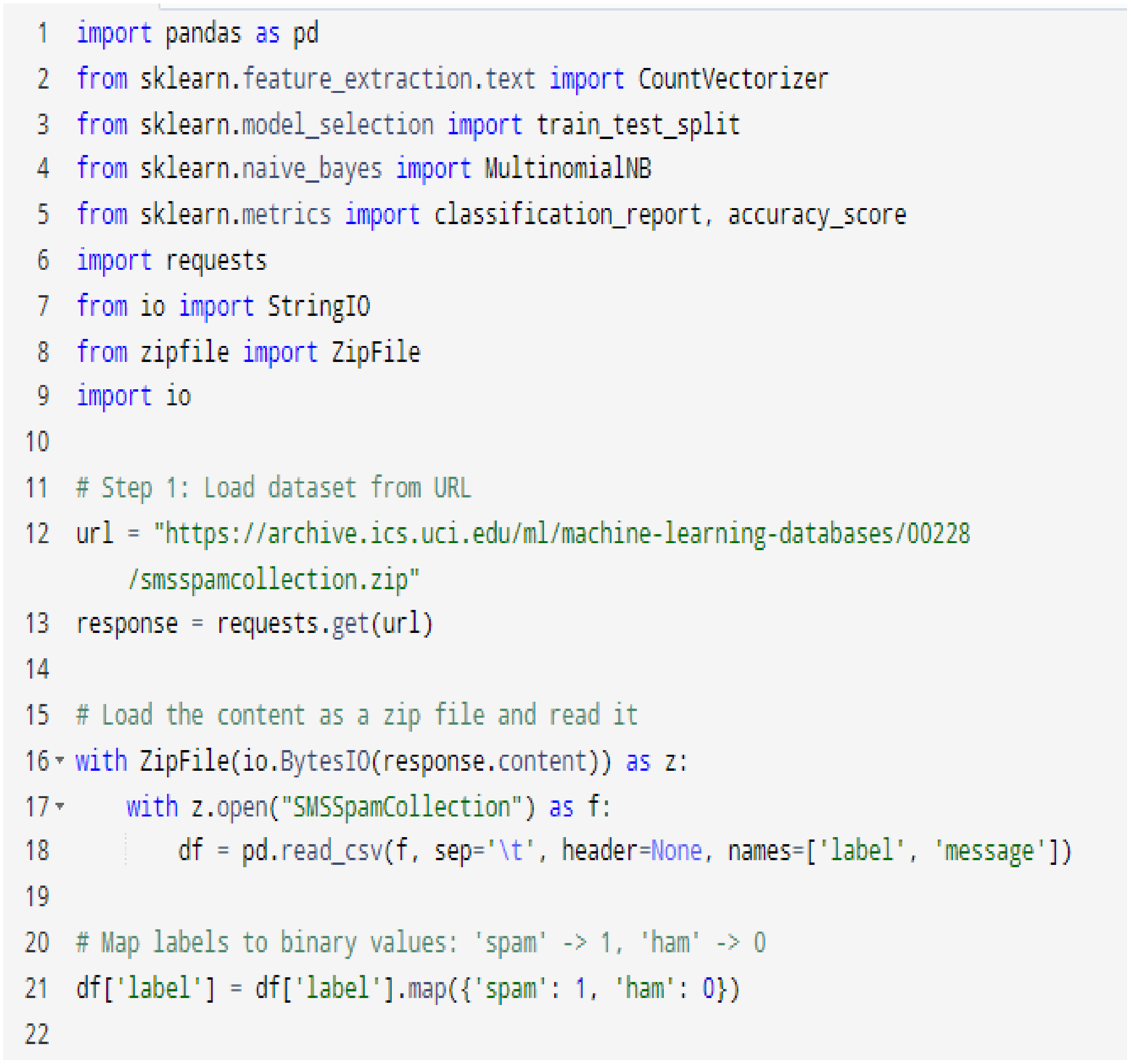
Once the model is trained and evaluated, it can be deployed for real-time spam classification. Users can input text (email or SMS), and the model will predict whether the message is spam or ham, along with the probability.

```
# User Input for Spam Prediction with Probability Output
user_input = input("Enter a message to check if it's spam (or type 'exit' to quit): ")
```

In a production environment, this could be integrated into email servers or messaging platforms to filter spam in real-time.

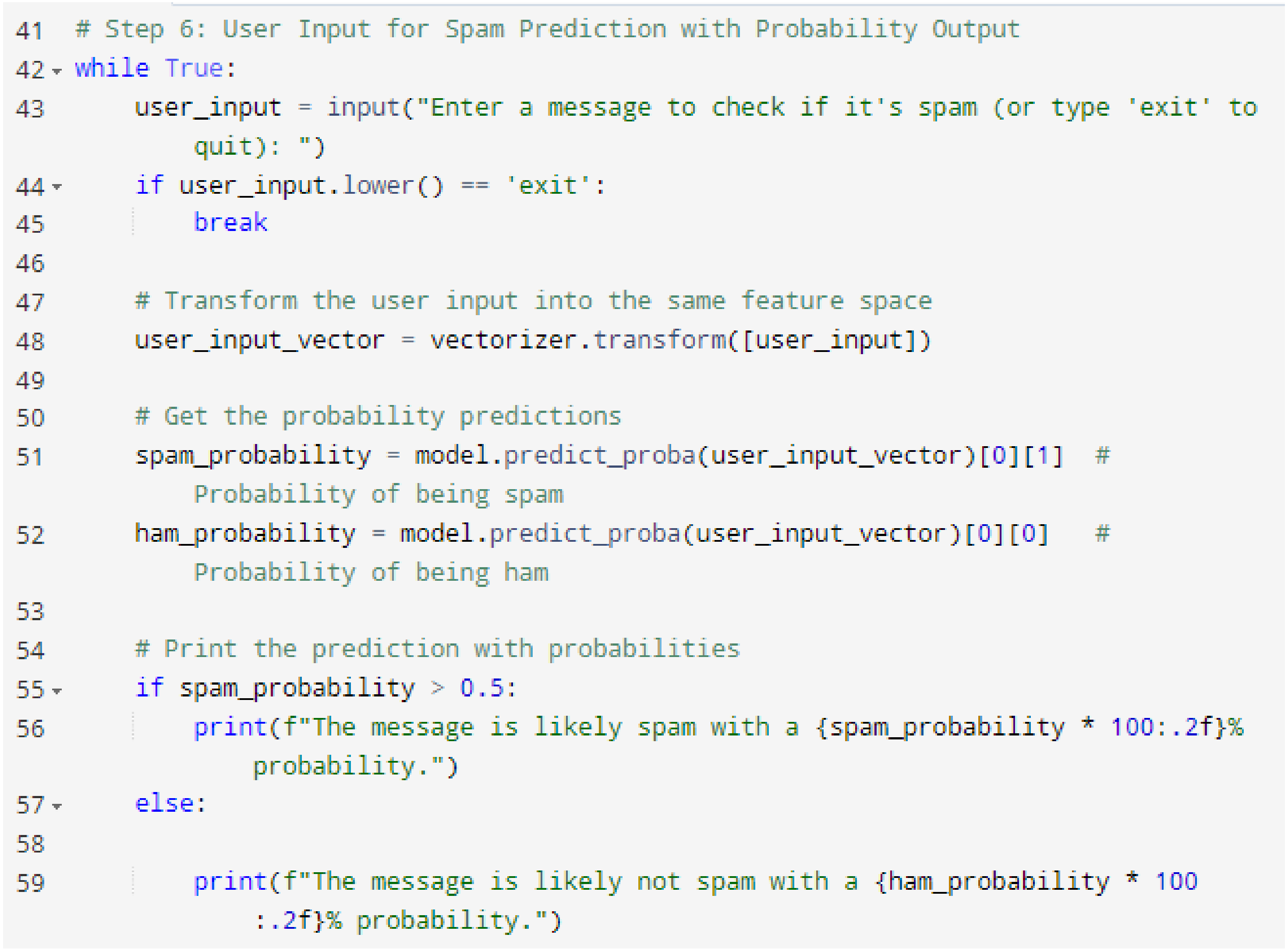
7. Challenges and Considerations

- **Data Quality:** The quality and size of the dataset directly impact the model's performance. Smaller or noisy datasets may lead to overfitting or poor generalization.
- **Model Overfitting:** Ensuring the model generalizes well on unseen data by tuning hyperparameters and using techniques like cross-validation.
- **Real-time Latency:** For real-time spam filtering, the model must make predictions quickly without significant delays.
- **Handling New Spam Techniques:** Spam messages evolve over time, so the model must be periodically retrained with new data to handle emerging spam techniques.



```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
import requests
from io import StringIO
from zipfile import ZipFile
import io

# Step 1: Load dataset from URL
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspamcollection.zip"
response = requests.get(url)

# Load the content as a zip file and read it
with ZipFile(io.BytesIO(response.content)) as z:
    with z.open("SMSSpamCollection") as f:
        df = pd.read_csv(f, sep='\t', header=None, names=['label', 'message'])

# Map labels to binary values: 'spam' -> 1, 'ham' -> 0
df['label'] = df['label'].map({'spam': 1, 'ham': 0})

```

```python
# Step 2: Text vectorization using CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['message'])  # Transform text data into feature vectors
y = df['label']

# Step 3: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 4: Train a Naive Bayes classifier
model = MultinomialNB()
model.fit(X_train, y_train)

# Step 5: Make predictions on the test set and evaluate the model
y_pred = model.predict(X_test)
print("Model Training Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

```



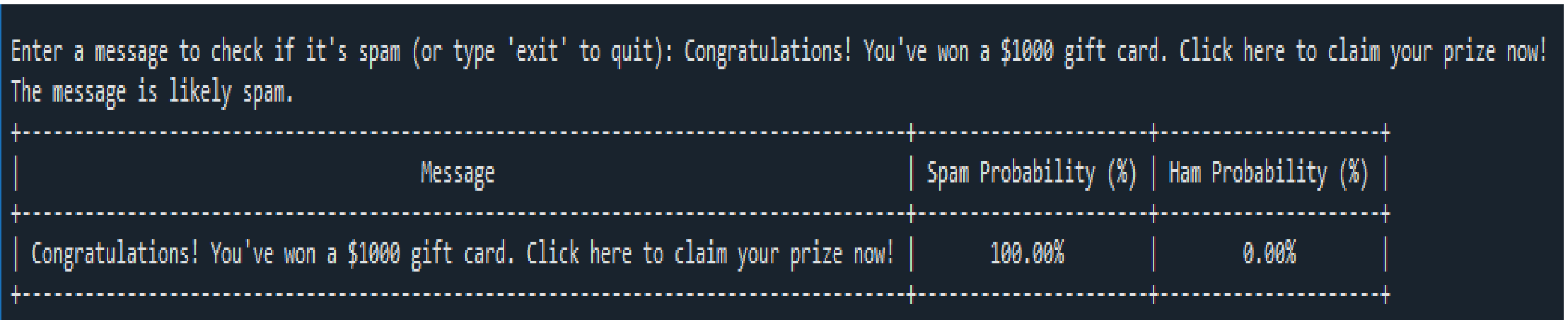
```python
# Step 6: User Input for Spam Prediction with Probability Output
while True:
    user_input = input("Enter a message to check if it's spam (or type 'exit' to quit): ")
    if user_input.lower() == 'exit':
        break

    # Transform the user input into the same feature space
    user_input_vector = vectorizer.transform([user_input])

    # Get the probability predictions
    spam_probability = model.predict_proba(user_input_vector)[0][1]  # Probability of being spam
    ham_probability = model.predict_proba(user_input_vector)[0][0]   # Probability of being ham

    # Print the prediction with probabilities
    if spam_probability > 0.5:
        print(f"The message is likely spam with a {spam_probability * 100:.2f}% probability.")
    else:

        print(f"The message is likely not spam with a {ham_probability * 100:.2f}% probability.")
```

Spam Example:

Message:
"Congratulations! You've won a $1000 gift card. Click here to claim your prize now!"

- **Reason**: This message is a typical **spam** message because it contains an unsolicited offer, often linked to phishing or other scams. It aims to get the user to click on a link to steal personal information or install malware.

Output:



```
Enter a message to check if it's spam (or type 'exit' to quit): Congratulations! You've won a $1000 gift card. Click here to claim your prize now!
The message is likely spam.
+-----------------------------------------------------------------------------+---------------------+---------------------+
|                                   Message                                   | Spam Probability (%) | Ham Probability (%) |
+-----------------------------------------------------------------------------+---------------------+---------------------+
| Congratulations! You've won a $1000 gift card. Click here to claim your prize now! |       100.00%       |        0.00%        |
+-----------------------------------------------------------------------------+---------------------+---------------------+
```

Message:

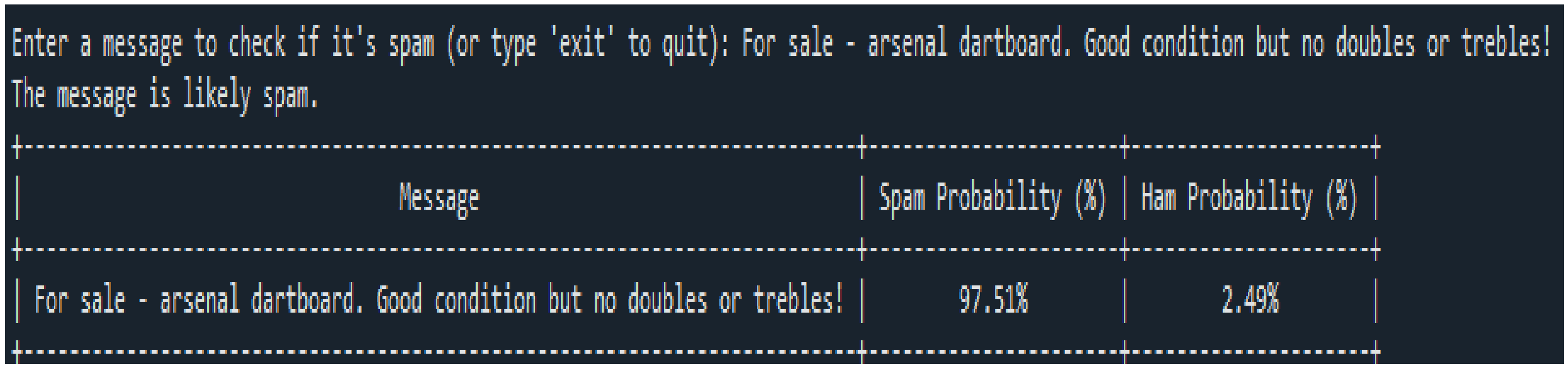
For sale - arsenal dartboard. Good condition but no doubles or trebles!

Why it's spam: Promotional Nature: The use of the phrase "For sale"

suggests an advertisement or offer to sell something.

Pattern Recognition: The classifier has learned that similar phrases (like "For sale") are often associated with spam, especially in the context of unsolicited messages about products or services.

Output:



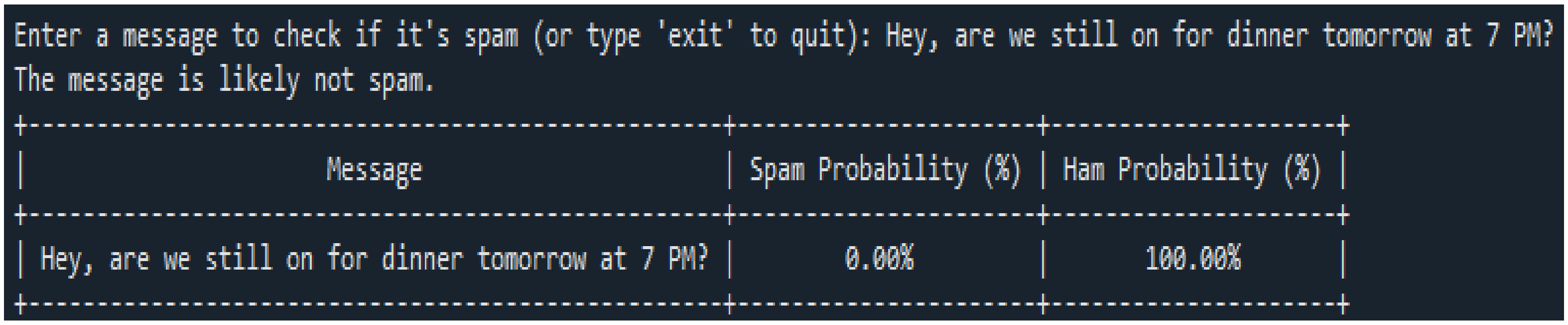
```
Enter a message to check if it's spam (or type 'exit' to quit): For sale - arsenal dartboard. Good condition but no doubles or trebles!
The message is likely spam.
+------------------------------------------------------------------------+---------------------+---------------------+
|                                 Message                                | Spam Probability (%) | Ham Probability (%) |
+------------------------------------------------------------------------+---------------------+---------------------+
| For sale - arsenal dartboard. Good condition but no doubles or trebles! |        97.51%       |        2.49%        |
+------------------------------------------------------------------------+---------------------+---------------------+
```

Non-Spam (Ham) Example:

Message:
"Hey, are we still on for dinner tomorrow at 7 PM?"

- **Reason**: This message is **non-spam** (ham) because it is a regular, personal communication between individuals. It is legitimate and not intended for mass distribution or to deceive the recipient.

Output:



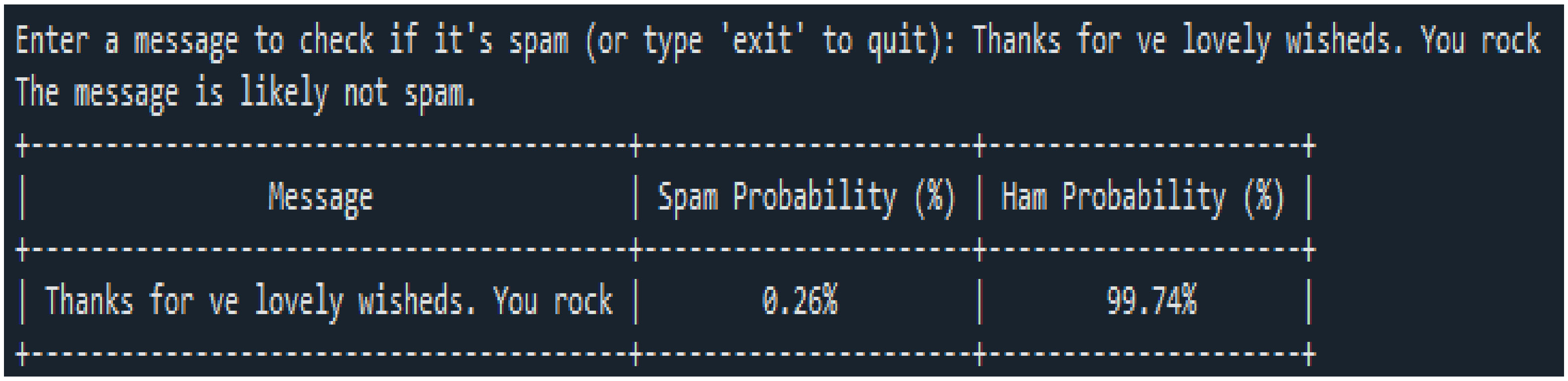
```
Enter a message to check if it's spam (or type 'exit' to quit): Hey, are we still on for dinner tomorrow at 7 PM?
The message is likely not spam.
+--------------------------------------------------+---------------------+--------------------+
|                      Message                     | Spam Probability (%) | Ham Probability (%) |
+--------------------------------------------------+---------------------+--------------------+
| Hey, are we still on for dinner tomorrow at 7 PM? |        0.00%        |      100.00%       |
+--------------------------------------------------+---------------------+--------------------+
```

Message:

Thanks for ve lovely wisheds. You rock

Why it's non-spam: The message "Thanks for ve lovely wisheds. You rock" is classified as ham because it has a personal and friendly tone, lacks any promotional language or unsolicited offers, and appears to be part of a genuine, positive interaction

Output:



```
Enter a message to check if it's spam (or type 'exit' to quit): Thanks for ve lovely wisheds. You rock
The message is likely not spam.
+----------------------------------------+----------------------+---------------------+
|                 Message                | Spam Probability (%) | Ham Probability (%) |
+----------------------------------------+----------------------+---------------------+
| Thanks for ve lovely wisheds. You rock |        0.26%         |       99.74%        |
+----------------------------------------+----------------------+---------------------+
```

Conclusion

The Spam Detection AI model developed here demonstrates the ability to classify SMS messages as either spam or ham using machine learning. The process involves preprocessing text data, extracting features using CountVectorizer, and training a Naive Bayes classifier. The system can be deployed for real-time filtering and integrated into email or SMS services. While the model performs well, addressing challenges such as imbalanced datasets and model retraining will improve long-term performance and scalability.

```
# Step 4: Train a Naive Bayes classifier
model = MultinomialNB()
model.fit(X_train, y_train)

# Step 5: Make predictions on the test set and evaluate the model
y_pred = model.predict(X_test)
print("Model Training Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Step 6: User Input for Spam Prediction with Probability Output
while True:
    user_input = input("Enter a message to check if it's spam (or type 'exit' to quit): ")
    if user_input.lower() == 'exit':
        break

    # Transform the user input into the same feature space
    user_input_vector = vectorizer.transform([user_input])

    # Get the probability predictions
    spam_probability = model.predict_proba(user_input_vector)[0][1]  # Probability of being spam
    ham_probability = model.predict_proba(user_input_vector)[0][0]   # Probability of being ham

    # Print the prediction with probabilities
    if spam_probability > 0.5:
        print(f"The message is likely spam with a {spam_probability * 100:.2f}% probability.")
    else:

        print(f"The message is likely not spam with a {ham_probability * 100:.2f}% probability.")
```

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
import requests
from io import StringIO
from zipfile import ZipFile
import io

# Step 1: Load dataset from URL
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00228/smsspamcollection.zip"
response = requests.get(url)

# Load the content as a zip file and read it
with ZipFile(io.BytesIO(response.content)) as z:
    with z.open("SMSSpamCollection") as f:
        df = pd.read_csv(f, sep='\t', header=None, names=['label', 'message'])

# Map labels to binary values: 'spam' -> 1, 'ham' -> 0
df['label'] = df['label'].map({'spam': 1, 'ham': 0})

# Step 2: Text vectorization using CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['message'])  # Transform text data into feature vectors
y = df['label']

# Step 3: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```