TEXT CLASSIFICATION

Presented by Group 1
Rishita Dey (23EC8028)
Astha Singh(23EC8079)
Dandolu Lakshmi Deepika (23EC8088)

TABLE OF CONTENTS

OBJECTIVE

- The main goal of this project is to classify SMS messages as either Spam or Ham (non-spam).
- Both classical machine learning algorithms and a deep learning model are implemented to compare performance.

DATASET OVERVIEW

Attribute	Description
Dataset Name	spam.csv
Total Samples	5572
Classes	ham (4825), spam (747)
Ham Percentage	86.59%
Spam Percentage	13.41%
Columns Used	v1 (label), v2 (message text)
Data Type	Text classification dataset
Problem Type	Binary Classification

IMPORTING REQUIRED LIBRARIES

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
import nltk, re, collections, pickle, os # nltk - Natural Language Toolkit
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Embedding, LSTM, Dropout, Bidirectional
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
```

- The dataset is loaded from a CSV file containing SMS messages labeled as spam or ham.
- Unnecessary columns are removed, column names are standardized, and duplicate entries are dropped.
- Finally, the dataset is checked for any missing values to ensure clean and reliable data for further processing.

```
df_spam = pd.read_csv('spam.csv', encoding = 'latin-1')
```

```
df_spam = df_spam.filter(['v1', 'v2'], axis = 1)
df_spam.columns = ['feature', 'message']
df_spam.drop_duplicates(inplace = True, ignore_index = True)
print('Number of null values:\n')
df_spam.isnull().sum()
```

```
Number of null values:
```

	0
feature	0
message	0

dtype: int64

- The dataset consists of two columns — feature (label) and message (text).
- It contains 5169 entries, with 4516 ham and 653 spam messages.

```
df_spam['feature'].value_counts()
```

feature	count
ham	4516
spam	653

dtype: int64

```
df_spam.shape, df_spam.columns
```

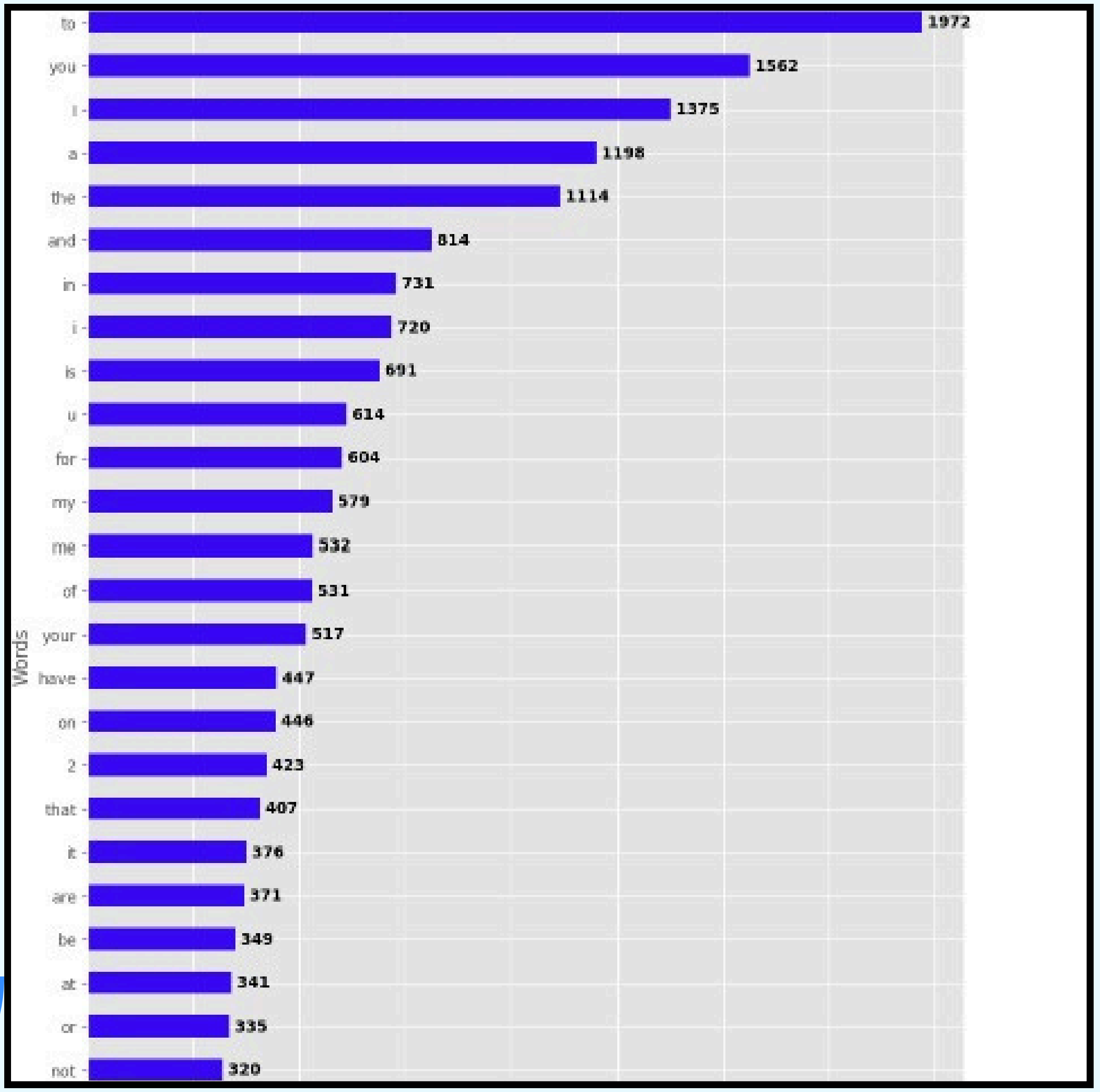
```
((5169, 2), Index(['feature', 'message'], dtype='object'))
```

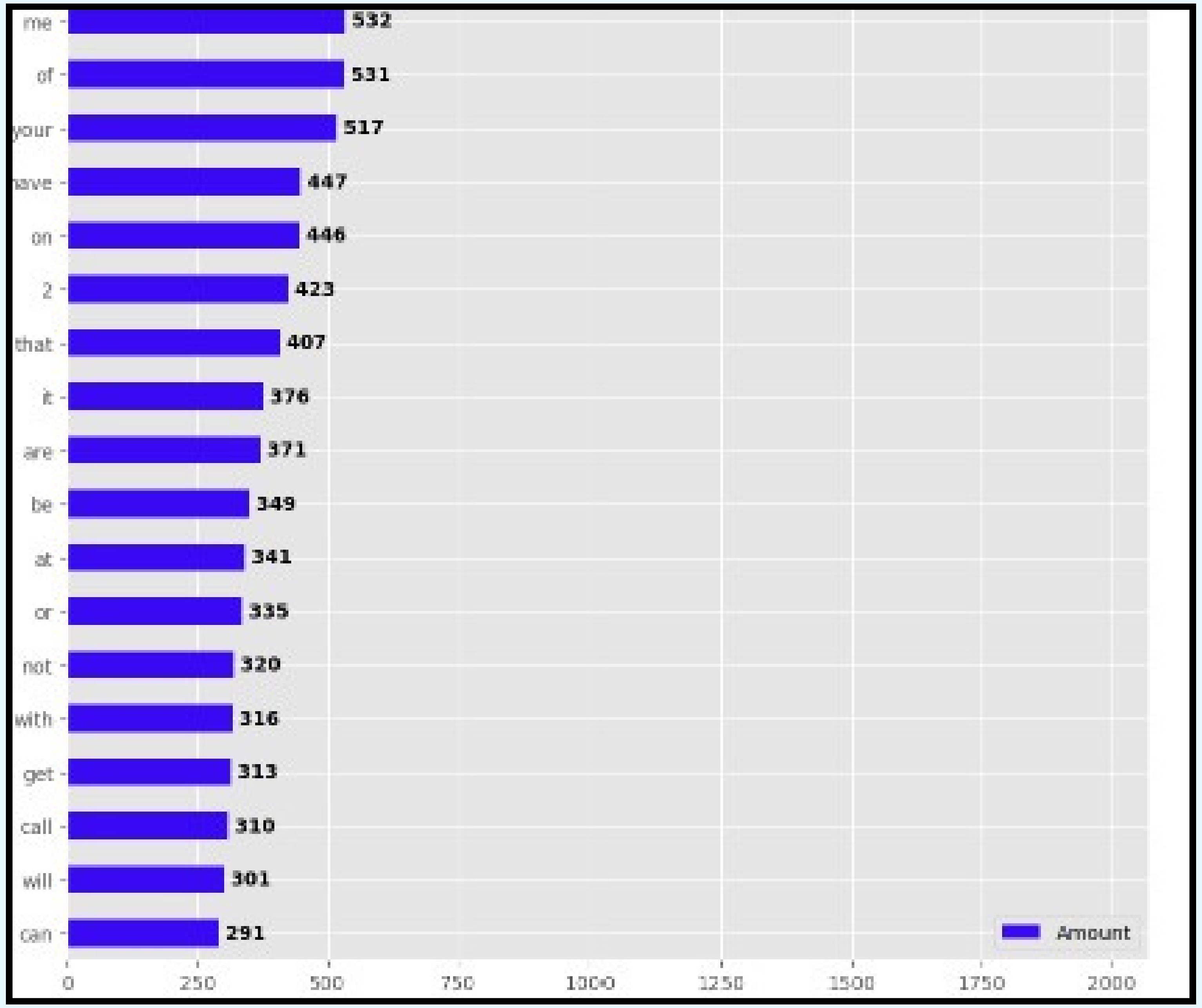
```
df_spam.describe().T
```

	count	unique	top	freq
feature	5169	2	ham	4516
message	5169	5169	Rofl. Its true to its name	1

THE FOLLOWING TABLE SHOWS THE FREQUENCY OF WORDS IN THE DATASET BEFORE PREPROCESSING

```
plot_words(df_spam['message'], number = 30)
```





PREPROCESSING STEP FOR CLASSICAL ML MODELS

```
print("\t\tStage I. Preliminary actions. Preparing of needed sets\n")
full_df_l = []
lemmatizer = WordNetLemmatizer()
for i in range(df_spam.shape[0]):
    mess_1 = df_spam.iloc[i, 1]
    mess_1 = re.sub('\b[\w\-.]+?@\w+?\.\w{2,4}\b', 'emailaddr', mess_1)
    mess_1 = re.sub('(http[s]?\S+)|(\w+\.[A-Za-z]{2,4}\S*)', 'httpaddr', mess_1)
    mess_1 = re.sub('£|\$', 'moneysymb', mess_1)
    mess_1 = re.sub('\b(\+\d{1,2}\s)?\d?[\-(.]?\d{3}\)?[\s.-]?\d{3}[\s.-]?\d{4}\b', 'phonenumbr', mess_1)
    mess_1 = re.sub('\d+(\.\d+)?', 'numbr', mess_1)
    mess_1 = re.sub('[^\w\d\s]', ' ', mess_1)
    mess_1 = re.sub('[^A-Za-z]', ' ', mess_1).lower()
    token_messages = word_tokenize(mess_1)
    mess = []
    for word in token_messages:
        if word not in set(stopwords.words('english')):
            mess.append(lemmatizer.lemmatize(word))
    txt_mess = " ".join(mess)
    full_df_l.append(txt_mess)
```

```
Stage I. Preliminary actions. Preparing of needed sets
```

🧹 Text Cleaning

Dropped duplicate rows and unnecessary columns.

Converted all text to lowercase.

Replaced specific text patterns:

Emails → emailaddr

URLs → httpaddr

Currency symbols → moneysymb

Phone numbers → phonenumbr

Numbers → numbr

Removed non-alphabetic characters (punctuation, digits, special symbols).

✂ Tokenization and Stopword Removal

Split text into individual words (tokens).

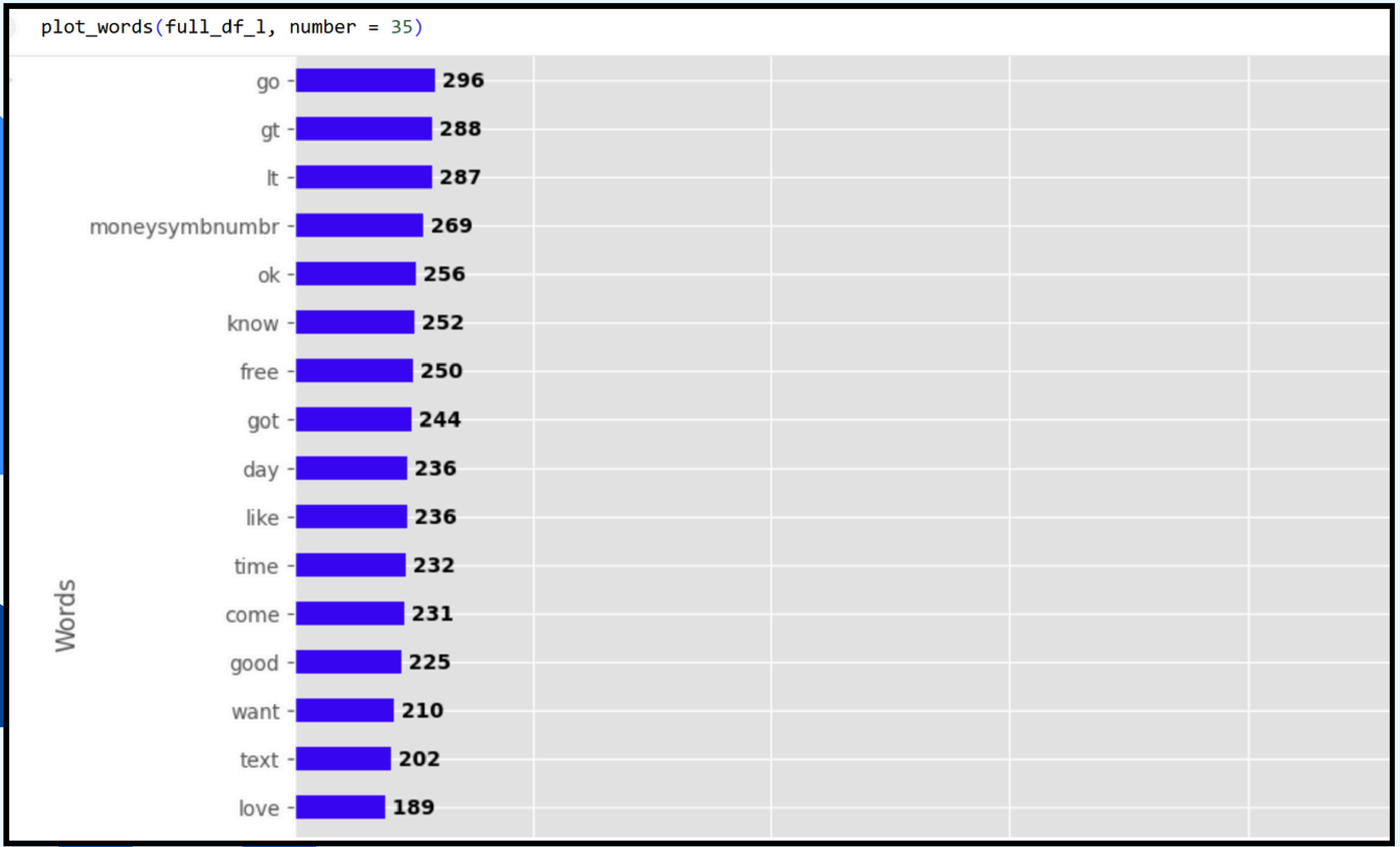
Removed common stopwords (like "the", "and", "is") using NLTK's English stopword list.

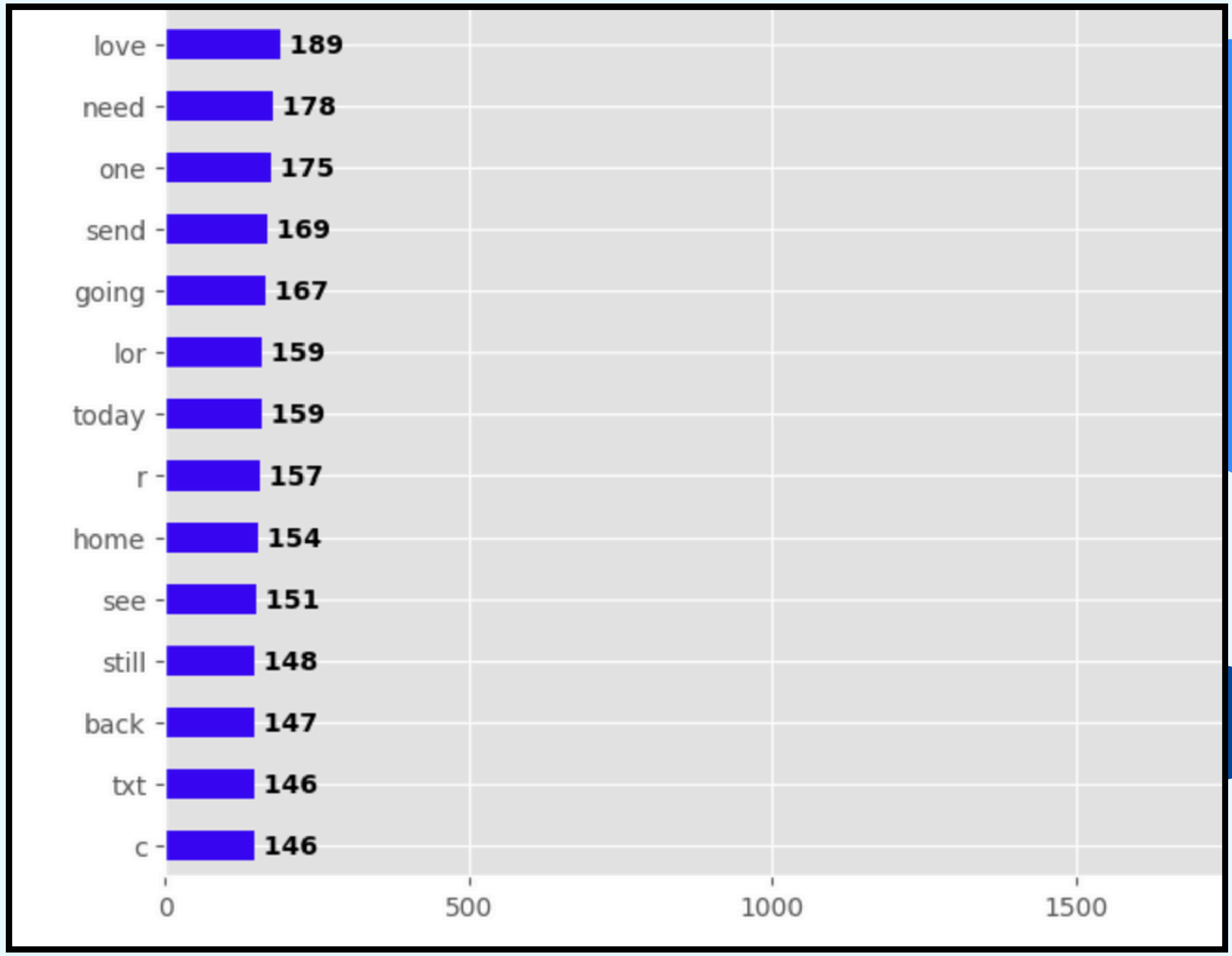
🧠 Lemmatization

Applied WordNetLemmatizer to reduce words to their base form.

Example: "running", "ran", → "run".

THE FOLLOWING TABLE SHOWS THE FREQUENCY OF WORDS IN THE DATASET AFTER PREPROCESSING





FEATURE EXTRACTION STEPS FOR CLASSICAL ML MODELS

```
add_df = CountVectorizer(max_features = size_vocabulary)
X = add_df.fit_transform(full_df_l).toarray()
y = df_spam.iloc[:, 0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = (test_size + valid_size), random_state = seed)
print('Number of rows in test set: ' + str(X_test.shape))
print('Number of rows in training set: ' + str(X_train.shape))
```

```
Number of rows in test set: (1293, 1000)
Number of rows in training set: (3876, 1000)
```

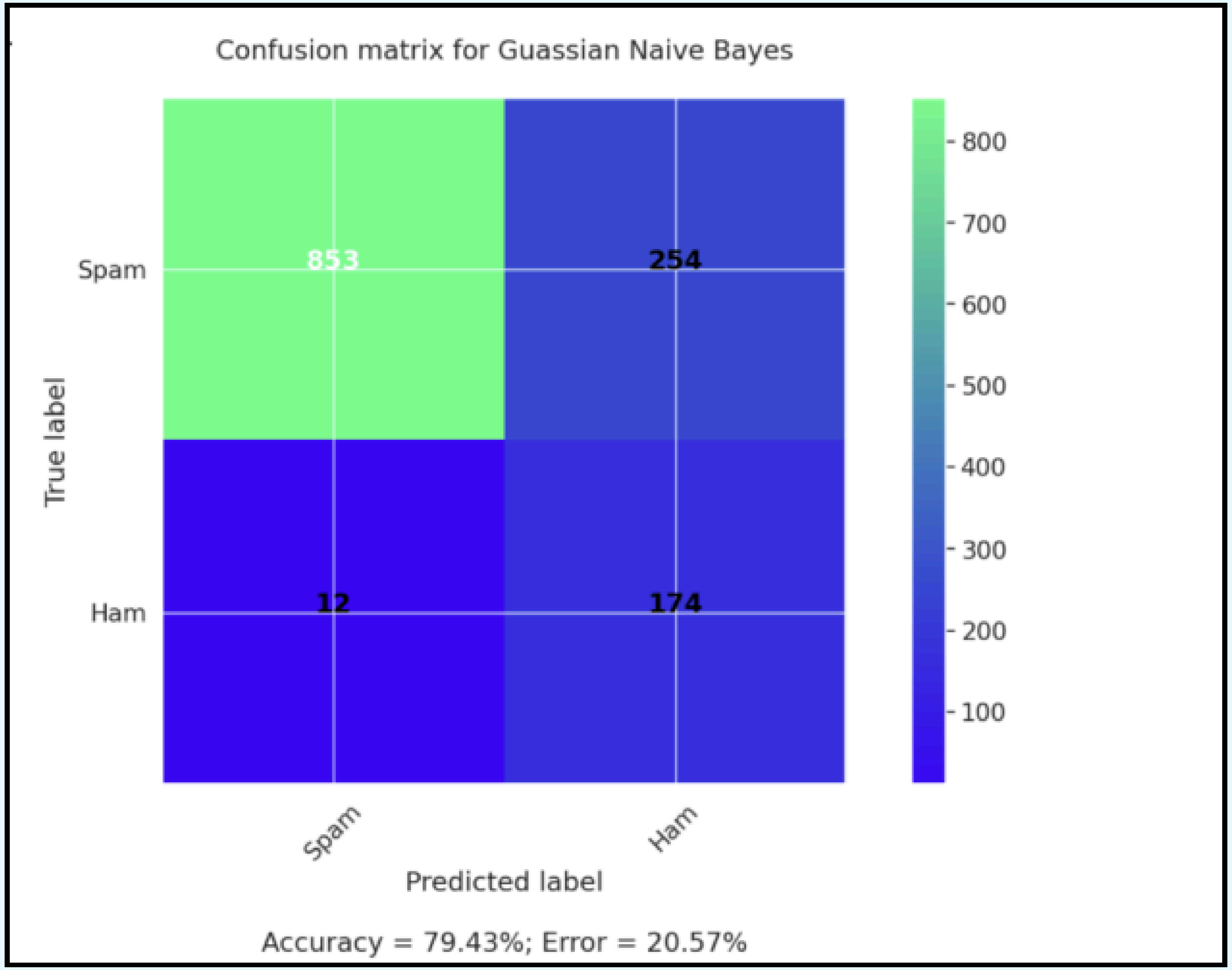
PERFORMANCE ANALYSIS OF CLASSICAL MODELS

1) GAUSSIAN NAIVE BAYES

```
print("\t\tStage IIa. Guassian Naive Bayes\n")
class_NBC = GaussianNB().fit(X_train, y_train) # Guassian Naive Bayes
y_pred_NBC = class_NBC.predict(X_test)
print('The first two predicted labels:', y_pred_NBC[0],y_pred_NBC[1], '\n')
conf_m_NBC = confusion_matrix(y_test, y_pred_NBC)
class_rep_NBC = classification_report(y_test, y_pred_NBC)
print('\t\t\tClassification report:\n\n', class_rep_NBC, '\n')
plot_conf_matr(conf_m_NBC, classes = ['Spam','Ham'], normalize = False, title = 'Confusion matrix for Guassian Naive Bayes')
```

```
          Stage IIa. Guassian Naive Bayes

 The first two predicted labels: spam ham

              Classification report:

               precision    recall  f1-score   support

         ham       0.99      0.77      0.87      1107
        spam       0.41      0.94      0.57       186

    accuracy                           0.79      1293
   macro avg       0.70      0.85      0.72      1293
weighted avg       0.90      0.79      0.82      1293
```

CONFUSION MATRIX FOR GAUSSIAN NAIVE BAYES MODEL

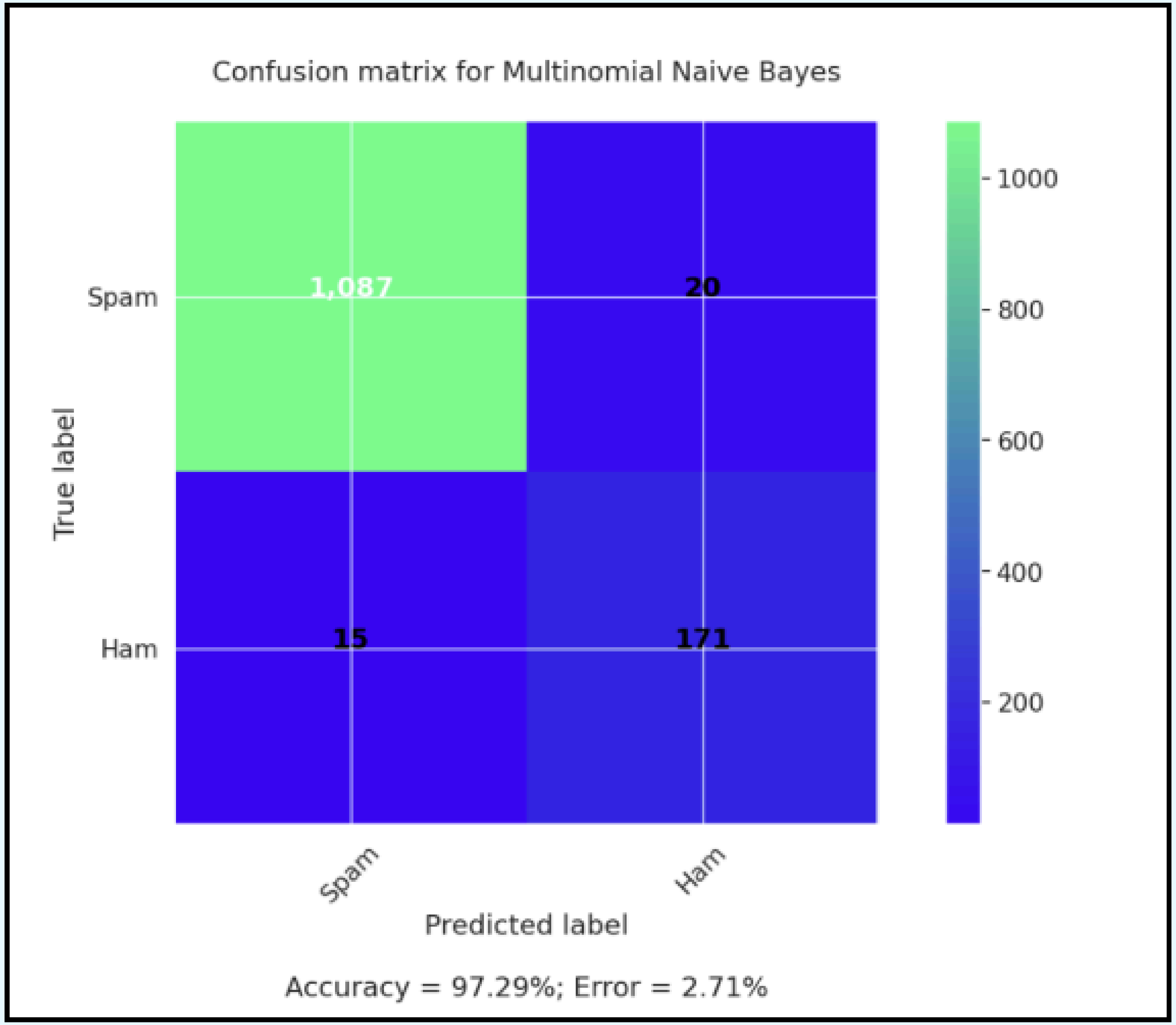


2) MULTINOMIAL NAIVE BAYES

```
print("\t\tStage IIb. Multinomial Naive Bayes\n")
class_MNB = MultinomialNB().fit(X_train, y_train) # Multinomial Naive Bayes
y_pred_MNB = class_MNB.predict(X_test)
print('The first two predicted labels:', y_pred_MNB[0],y_pred_MNB[1], '\n')
conf_m_MNB = confusion_matrix(y_test, y_pred_MNB)
class_rep_MNB = classification_report(y_test, y_pred_MNB)
print('\t\t\tClassification report:\n\n', class_rep_MNB, '\n')
plot_conf_matr(conf_m_MNB, classes = ['Spam','Ham'], normalize = False, title = 'Confusion matrix for Multinomial Naive Bayes')
```

```
            Stage IIb. Multinomial Naive Bayes

 The first two predicted labels: ham ham

                        Classification report:

               precision    recall  f1-score   support

         ham       0.99      0.98      0.98      1107
        spam       0.90      0.92      0.91       186

    accuracy                           0.97      1293
   macro avg       0.94      0.95      0.95      1293
weighted avg       0.97      0.97      0.97      1293
```

CONFUSION MATRIX FOR MULTINOMIAL NAIVE BAYES MODEL

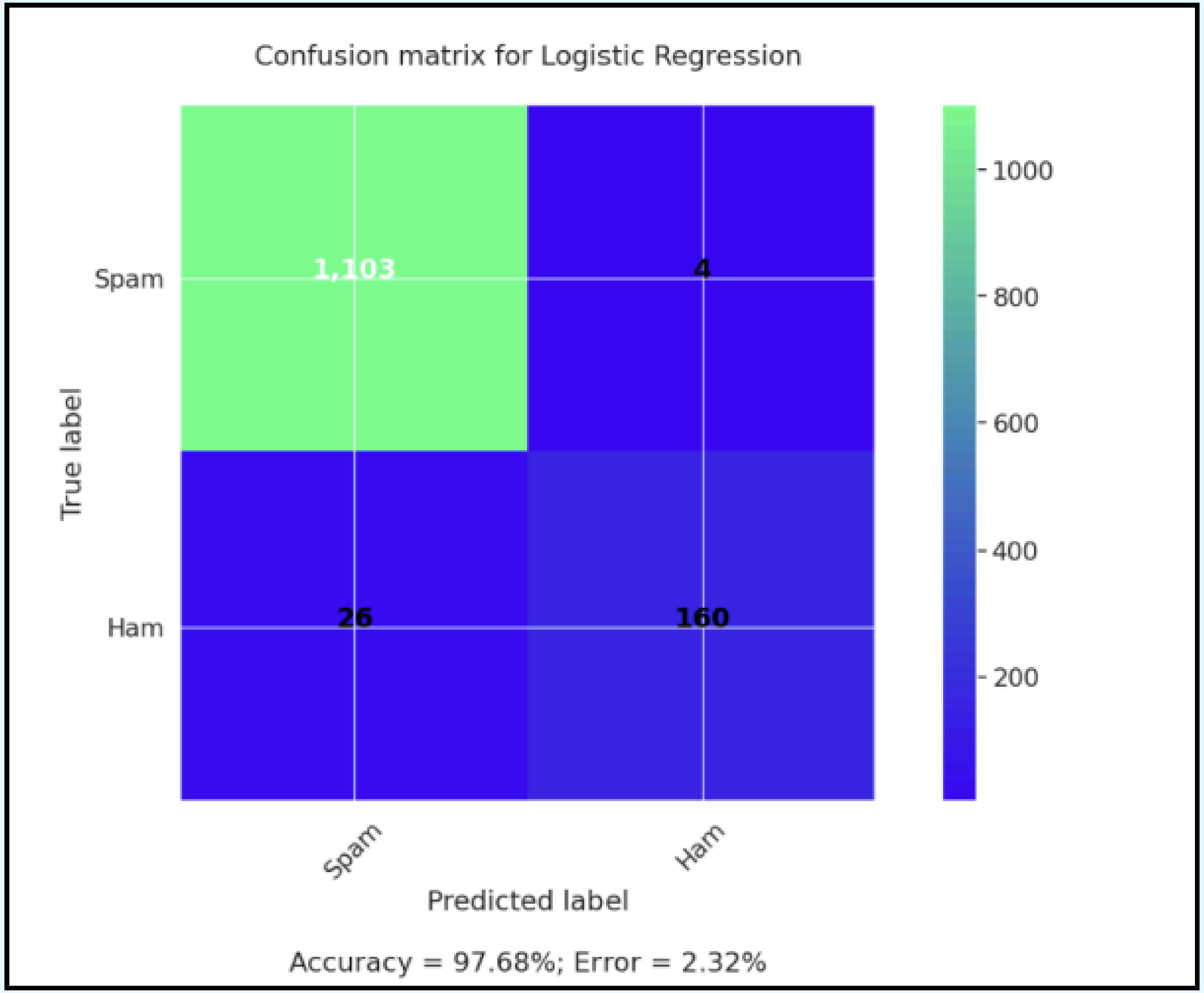


3) LOGISTIC REGRESSION

```
print("\t\tStage IV. Logistic Regression\n")
class_LR = LogisticRegression(random_state = seed, solver = 'liblinear').fit(X_train, y_train)
y_pred_LR = class_LR.predict(X_test)
print('The first two predicted labels:', y_pred_LR[0], y_pred_LR[1], '\n')
conf_m_LR = confusion_matrix(y_test, y_pred_LR)
class_rep_LR = classification_report(y_test, y_pred_LR)
print('\t\t\tClassification report:\n\n', class_rep_LR, '\n')
plot_conf_matr(conf_m_LR, classes = ['Spam','Ham'], normalize = False, title = 'Confusion matrix for Logistic Regression')
```

```
        Stage IV. Logistic Regression

The first two predicted labels: ham ham

             Classification report:

              precision    recall  f1-score   support

         ham       0.98      1.00      0.99      1107
        spam       0.98      0.86      0.91       186

    accuracy                           0.98      1293
   macro avg       0.98      0.93      0.95      1293
weighted avg       0.98      0.98      0.98      1293
```

CONFUSION MATRIX FOR LOGISTIC REGRESSION MODEL



DEEP LEARNING

PRE PROCESSING STEPS FOR DL MODEL

```
print("Stage I. Preliminary actions. Preparing of needed sets\n")

sentences_new_set = []
labels_new_set = []
for i in range(0, df_spam.shape[0], 1):
    sentences_new_set.append(df_spam['message'][i])
    labels_new_set.append(df_spam['feature'][i])
```

```
Stage I. Preliminary actions. Preparing of needed sets
```

```
train_size = int(df_spam.shape[0] * (1 - test_size - valid_size))
valid_bound = int(df_spam.shape[0] * (1 - valid_size))

train_sentences = sentences_new_set[0 : train_size]
valid_sentences = sentences_new_set[train_size : valid_bound]
test_sentences = sentences_new_set[valid_bound : ]

train_labels_str = labels_new_set[0 : train_size]
valid_labels_str = labels_new_set[train_size : valid_bound]
test_labels_str = labels_new_set[valid_bound : ]
```

PRE PROCESSING STEPS FOR DL MODEL

```
print("Stage II. Labels transformations\n")

train_labels = [0] * len(train_labels_str)
for ind, item in enumerate(train_labels_str):
    if item == 'ham':
        train_labels[ind] = 1
    else:
        train_labels[ind] = 0

valid_labels = [0] * len(valid_labels_str)
for ind, item in enumerate(valid_labels_str):
    if item == 'ham':
        valid_labels[ind] = 1
    else:
        valid_labels[ind] = 0

test_labels = [0] * len(test_labels_str)
for ind, item in enumerate(test_labels_str):
    if item == 'ham':
        test_labels[ind] = 1
    else:
        test_labels[ind] = 0

train_labels = np.array(train_labels)
valid_labels = np.array(valid_labels)
test_labels = np.array(test_labels)
```

- Preprocessing prepares the raw dataset for deep learning.
- The data is divided into training, validation, and test sets.
- Text messages and their corresponding labels are extracted and organized.
- Labels like “spam” and “ham” are converted into numeric values (1 and 0).
- This ensures the data is clean, consistent, and ready for model training and evaluation.

FEATURE EXTRACTION STEPS FOR DL MODEL

```
print("Stage III. Tokenization\n")

tokenizer = Tokenizer(num_words = size_vocabulary,
                      oov_token = oov_token,
                      lower = False)
tokenizer.fit_on_texts(train_sentences)
word_index = tokenizer.word_index
```

```
Stage III. Tokenization
```

```
train_sequences = tokenizer.texts_to_sequences(train_sentences)
size_voc = len(word_index) + 1
max_len = max([len(i) for i in train_sequences])
train_set = pad_sequences(train_sequences,
                                padding = padding_type,
                                maxlen = max_len,
                                truncating = trunc_type)

valid_sequences = tokenizer.texts_to_sequences(valid_sentences)
valid_set = pad_sequences(valid_sequences,
                               padding = padding_type,
                               maxlen = max_len,
                               truncating = trunc_type)

test_sequences = tokenizer.texts_to_sequences(test_sentences)
test_set = pad_sequences(test_sequences,
                               padding = padding_type,
                               maxlen = max_len,
                               truncating = trunc_type)
```

MODEL BUILDING (DEEP LEARNING)

```
print("Stage IV. Model building\n")

model = Sequential([
    Embedding(input_dim=size_voc, output_dim=embedding_dimension, input_shape=(max_len,)),
    Bidirectional(LSTM(100)),
    Dropout(drop_level),
    Dense(20, activation='relu'),
    Dropout(drop_level),
    Dense(1, activation='sigmoid')
])
```

- A Sequential Bidirectional LSTM model is created to capture text patterns in both directions.
- Embedding converts words into vector form, and Dropout prevents overfitting.
- Dense layers with ReLU and Sigmoid activations perform the final spam vs. ham classification.

MODEL COMPILING AND FITTING

```
print("Stage V. Model compiling & fitting\n")

optim = Adam(learning_rate=0.0003)
model.compile(loss='binary_crossentropy', optimizer=optim, metrics=['accuracy'])

model.summary()
```

```
Stage V. Model compiling & fitting
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 189, 64)	606,080
bidirectional (Bidirectional)	(None, 200)	132,000
dropout (Dropout)	(None, 200)	0
dense (Dense)	(None, 20)	4,020
dropout_1 (Dropout)	(None, 20)	0
dense_1 (Dense)	(None, 1)	21

Total params: 742,121 (2.83 MB)
Trainable params: 742,121 (2.83 MB)
Non-trainable params: 0 (0.00 B)

```
history = model.fit(train_set,
                    train_labels,
                    epochs = num_epochs,
                    validation_data = (valid_set, valid_labels),
                    verbose = 1)
```

- The model is set up using the Adam optimizer (learning rate = 0.0003) and binary crossentropy as the loss function.
- The summary shows the model layers, output shapes, and total 742,121 trainable parameters.
- The model is trained using the fit() function with training and validation data for several epochs to improve accuracy.

MODEL EVALUATION

```
model_score = model.evaluate(test_set, test_labels, batch_size = embedding_dimension, verbose = 1)
print(f"Test accuracy: {model_score[1] * 100:0.2f}% \t\t Test error: {model_score[0]:0.4f}")
```

```
17/17 ━━━━━━━━━━━━━━━━━━━━ 3s 169ms/step - accuracy: 0.9849 - loss: 0.0783
Test accuracy: 98.36%            Test error: 0.0771
```

- The trained model is tested on the test dataset using the evaluate() function.
- It calculates the accuracy and error (loss) to check how well the model performs on unseen data.
- The model achieved a test accuracy of 98.36% and a test error of 0.0771, showing excellent performance.

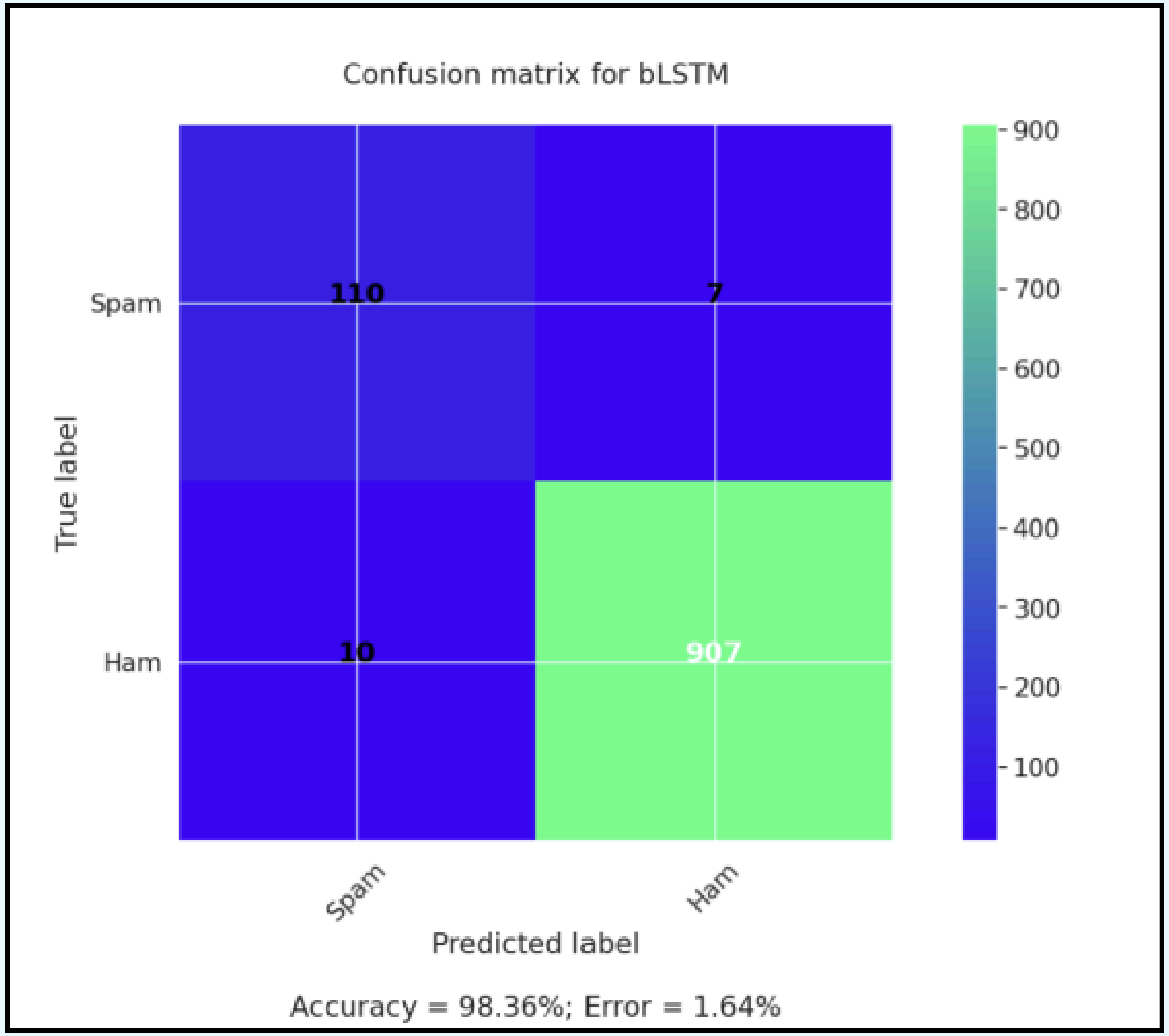
DEEP LEARNING MODEL PREDICTION AND PERFORMANCE ANALYSIS

BLSTM DEEP LEARNING MODEL

```
y_pred_bLSTM = model.predict(test_set)

y_prediction = [0] * y_pred_bLSTM.shape[0]
for ind, item in enumerate(y_pred_bLSTM):
    if item > threshold:
        y_prediction[ind] = 1
    else:
        y_prediction[ind] = 0

conf_m_bLSTM = confusion_matrix(test_labels, y_prediction)
class_rep_bLSTM = classification_report(test_labels, y_prediction)
print('\t\t\tClassification report:\n\n', class_rep_bLSTM, '\n')
plot_conf_matr(conf_m_bLSTM, classes = ['Spam','Ham'], normalize = False, title = 'Confusion matrix for bLSTM')
```

```
33/33 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step
			Classification report:

              precision    recall  f1-score   support

           0       0.92      0.94      0.93       117
           1       0.99      0.99      0.99       917

    accuracy                           0.98      1034
   macro avg       0.95      0.96      0.96      1034
weighted avg       0.98      0.98      0.98      1034
```

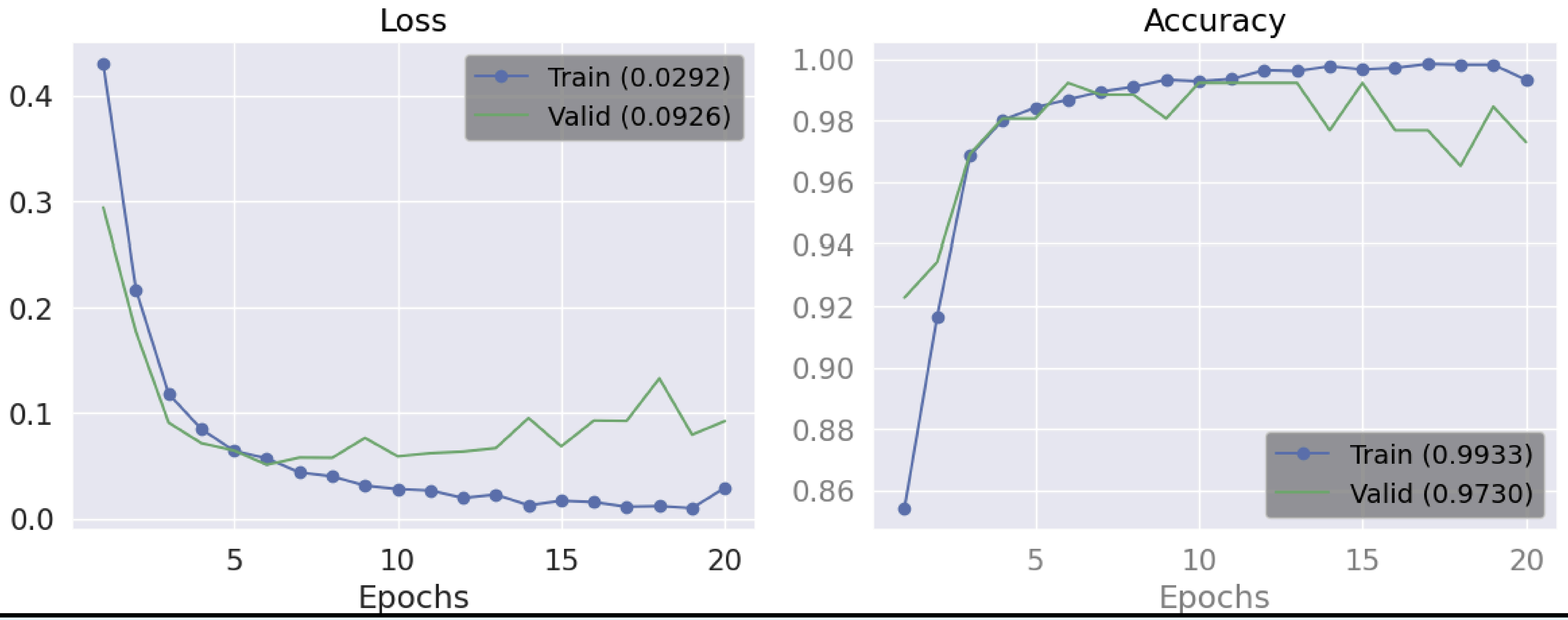
BLSTM CONFUSION MATRIX



RESULT VISUALIZATION

```
print("Stage VI. Results visualization\n")
plot_history(history)
```

```
Stage VI. Results visualization
```



📈 Performance Ranking

By Accuracy:

- Bidirectional LSTM & Logistic Regression: 98% ✅
- Multinomial Naive Bayes: 97% ✅
- Gaussian Naive Bayes: 79% ❌

By Spam Detection (Recall):

- Bidirectional LSTM: 99% 🥇
- Multinomial Naive Bayes: 92% 🥈
- Gaussian Naive Bayes: 94% 🥈 (but poor precision)
- Logistic Regression: 86% 🥉

By False Alarms (Ham Precision):

- Logistic Regression: 98% 🥇 (only 4 false alarms)
- Multinomial Naive Bayes: 99% 🥇 (but 20 false alarms)
- Bidirectional LSTM: 92% 🥈
- Gaussian Naive Bayes: 99% 🥇 (but terrible spam precision)

Final Conclusion

For a SMS spam filter system, it is recommended:

🥇 **First Choice: Logistic Regression**

- 98% accuracy with minimal false alarms
- Interpretable - it can be understood why decisions are made
- Computationally efficient - fast training and prediction
- Proven reliability - widely used in industry

🥈 **Excellent Alternative: Bidirectional LSTM**

- Best spam detection (99% recall)
- Future-proof - handles complex language patterns
- Context-aware - understands word sequences and relationships

THANK YOU