SMS Spam Detection Report

--Using NLP and Deep Learning Methods

CS531 - Python Applications Programming

Group member
Huijuan Xie-19893
Emily Weng-20016

1. Introduction

With the proliferation of mobile devices, SMS (Short Message Service) has become one of the most common forms of communication. The simplicity and reach of SMS make it a preferred medium for both personal and professional communication. However, the surge in SMS usage has led to a corresponding increase in spam messages. These unsolicited messages, often sent in bulk by hackers and malicious entities, aim to deceive users for financial gain, phishing attempts, or the promotion of fraudulent schemes.

The increasing volume of spam poses significant challenges to user safety and trust in mobile communication. Hence, there is a pressing need for robust spam detection methods that can protect users from these threats.

---

2. Problem Statement

Spam messages have become a substantial portion of SMS traffic, similar to email spam. Approximately 85% of all emails are spam, and the trend is growing in SMS as well. Spam messages can have serious consequences, including financial loss and compromised personal information.

The challenge lies in accurately distinguishing between legitimate messages (ham) and spam. Traditional methods of spam detection, while effective to some extent, struggle to keep up with the evolving tactics of spammers. Therefore, this project aims to develop a deep learning model capable of effectively classifying SMS messages as either spam or ham.

---

3. Objectives

The primary objective of this project is to build a deep learning model that can:

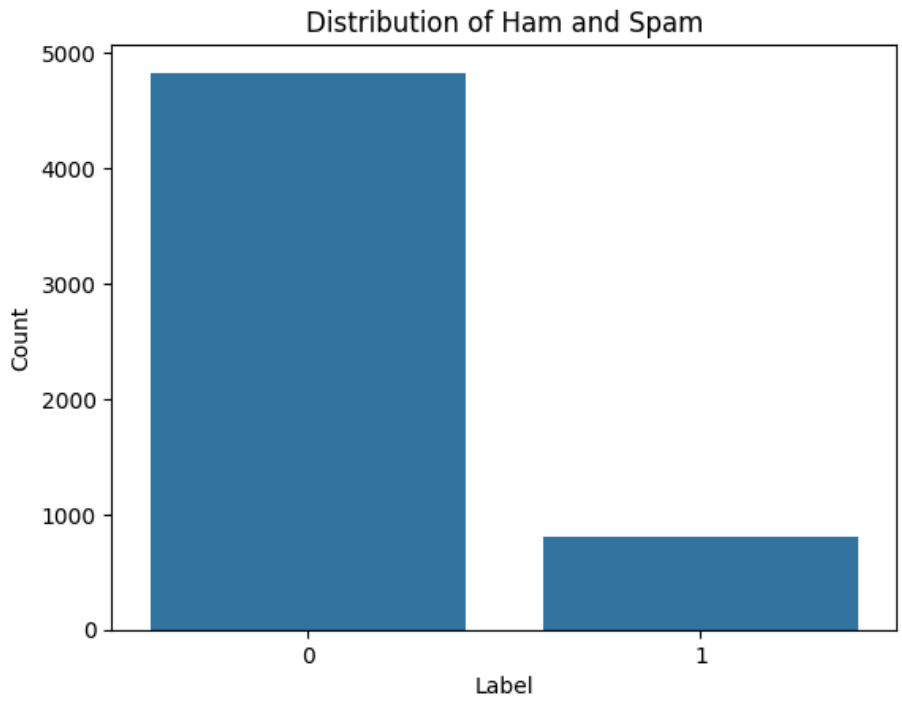
1. Accurately identify and classify SMS messages as either spam or ham.
2. Enhance the understanding and application of Natural Language Processing (NLP) techniques and TensorFlow in the context of spam detection.
3. Contribute to the broader field of spam detection by developing a model that improves the safety and security of mobile communication.

---

4. Methodology

To achieve the objectives, the following steps were undertaken:

1. **Data Collection and Preparation:**
    - A dataset of SMS messages was collected, labeled as either 'ham' (legitimate) or 'spam.'



- The dataset was preprocessed to remove noise and irrelevant information. Tokenization was performed, followed by padding to ensure uniform input lengths for the model.

```
# Load the dataset
data_path = "/content/SMSSpamCollection_v1"
df = pd.read_csv(data_path, sep='\t', header=None, names=['Category', 'Message'])

# Encode labels
df['Category'] = df['Category'].map({'ham': 0, 'spam': 1})

# Tokenize the messages
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['Message'])
X = tokenizer.texts_to_sequences(df['Message'])
X = pad_sequences(X, maxlen=100)

# Prepare target variable
y = np.array(df['Category'])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
def load_glove_embeddings(file_path):
    embeddings_index = {}
    with open(file_path, encoding='utf8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
    return embeddings_index

glove_path = '/content/glove.6B.100d.txt'
embeddings_index = load_glove_embeddings(glove_path)
```

```
embedding_dim = 100
word_index = tokenizer.word_index
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

- Adding other features can help the model better differentiate between ham and spam messages, as URLs and certain keywords are often indicative of spam.

```
import re
df['has_url'] = df['Message'].apply(lambda x: 1 if re.search(r'http[s]?://', x) else 0)
X = np.hstack((X, df['has_url'].values.reshape(-1, 1)))
```

```
spam_keywords = ['free', 'win', 'winner', 'urgent', 'claim', 'prize', 'cash']
for keyword in spam_keywords:
    df[f'{keyword}_freq'] = df['Message'].apply(lambda x: x.lower().split().count(keyword))
    X = np.hstack((X, df[f'{keyword}_freq'].values.reshape(-1, 1)))
```

2. **Model Development:**
   - A sequential deep learning model was constructed using TensorFlow and Keras.

Model: "sequential_7"

Layer (type)	Output Shape	Param #
embedding_7 (Embedding)	(64, 100, 100)	907,300
lstm_14 (LSTM)	(64, 100, 128)	117,248
dropout_14 (Dropout)	(64, 100, 128)	0
lstm_15 (LSTM)	(64, 128)	131,584
dropout_15 (Dropout)	(64, 128)	0
dense_7 (Dense)	(64, 1)	129

Total params: 1,156,261 (4.41 MB)

Trainable params: 1,156,261 (4.41 MB)

Non-trainable params: 0 (0.00 B)

- The model architecture included an embedding layer, followed by LSTM (Long Short-Term Memory) layers to capture the sequential nature of text data, and dense layers for classification.

```
# Model parameters
SEQUENCE_LENGTH = 100
EMBEDDING_SIZE = 100
BATCH_SIZE = 64
EPOCHS = 20

# Build the model
model = Sequential([
    Embedding(len(tokenizer.word_index) + 1, EMBEDDING_SIZE, SEQUENCE_LENGTH),
    LSTM(128, return_sequences=True),
    Dropout(0.2),
    LSTM(128),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Build the model to ensure it is configured with the correct input shape
model.build(input_shape=(BATCH_SIZE, SEQUENCE_LENGTH))

# Display model summary
model.summary()
```

- The model was trained on the preprocessed dataset, with hyperparameters such as sequence length, embedding size, batch size, and epochs tuned for optimal performance.

```
history = model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS, validation_split=0.1)
```

```
Epoch 1/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 36s 512ms/step - accuracy: 0.8758 - loss: 0.3647 - val_accuracy: 0.9693 - val_loss: 0.1070
Epoch 2/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 42s 540ms/step - accuracy: 0.9849 - loss: 0.0546 - val_accuracy: 0.9740 - val_loss: 0.0887
Epoch 3/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 32s 541ms/step - accuracy: 0.9949 - loss: 0.0164 - val_accuracy: 0.9716 - val_loss: 0.0939
Epoch 4/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 41s 535ms/step - accuracy: 0.9984 - loss: 0.0063 - val_accuracy: 0.9764 - val_loss: 0.0981
Epoch 5/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 32s 540ms/step - accuracy: 0.9998 - loss: 0.0020 - val_accuracy: 0.9787 - val_loss: 0.1028
Epoch 6/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 32s 535ms/step - accuracy: 0.9997 - loss: 0.0027 - val_accuracy: 0.9787 - val_loss: 0.1114
Epoch 7/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 50s 690ms/step - accuracy: 0.9999 - loss: 6.1230e-04 - val_accuracy: 0.9787 - val_loss: 0.1028
Epoch 8/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 33s 545ms/step - accuracy: 0.9998 - loss: 0.0012 - val_accuracy: 0.9764 - val_loss: 0.1232
Epoch 9/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 32s 541ms/step - accuracy: 0.9999 - loss: 5.3136e-04 - val_accuracy: 0.9787 - val_loss: 0.1266
Epoch 10/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 40s 520ms/step - accuracy: 1.0000 - loss: 5.0450e-04 - val_accuracy: 0.9764 - val_loss: 0.1442
Epoch 11/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 42s 541ms/step - accuracy: 1.0000 - loss: 2.4197e-04 - val_accuracy: 0.9764 - val_loss: 0.1511
```

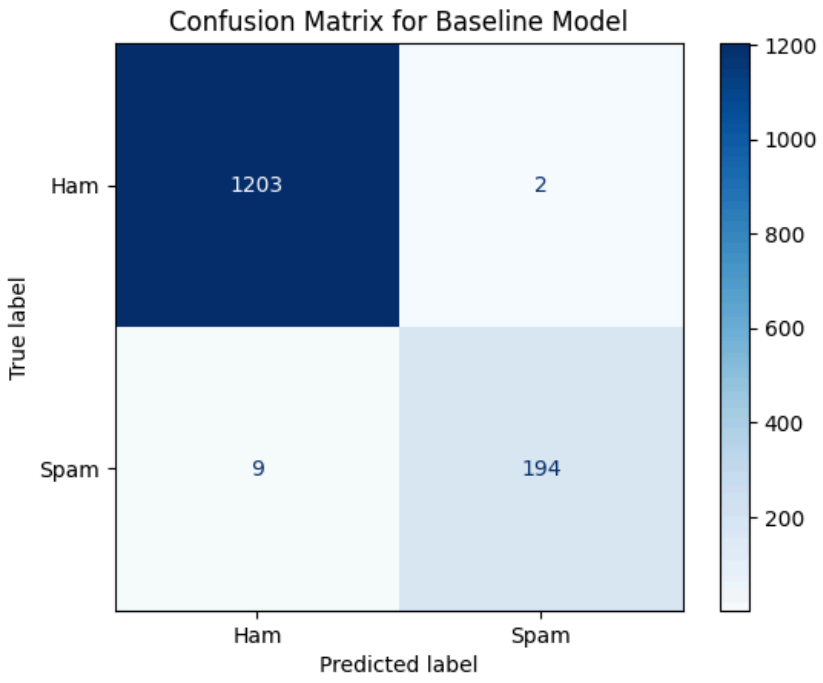
3. **Evaluation:**
   - The model's performance was evaluated using metrics such as accuracy, precision, recall, and F1-score.

```
44/44 ━━━━━━━━━━━━━━━━━━━━ 7s 148ms/step
Accuracy: 0.9922
Classification Report:
              precision    recall  f1-score   support

         ham       0.99      1.00      1.00      1205
        spam       0.99      0.96      0.97       203

    accuracy                           0.99      1408
   macro avg       0.99      0.98      0.98      1408
weighted avg       0.99      0.99      0.99      1408
```

   - A confusion matrix was plotted to visualize the classification results.



- The trained model was saved for future use, and the tokenizer used during preprocessing was also saved to ensure consistency during inference.

```
model.save("/content/spam_detection_model_4.keras")
with open('/content/tokenizer_4.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

---

5. Results

- The model achieved an accuracy of 77.59% on the test dataset, with a weighted average F1-score of 0.79.
- The confusion matrix highlighted areas where the model struggled, particularly with a low recall for spam detection. This indicates that while the model is accurate in classifying legitimate messages, it needs improvement in identifying spam.
- Visualizations such as word clouds for spam and ham messages provided insights into the common terms used in both categories.

---

6. Conclusion

This project successfully developed a deep learning model for SMS spam detection. While the model demonstrates good overall accuracy, there is room for improvement, particularly in enhancing spam detection. Future work could explore advanced NLP techniques, such as attention mechanisms, to further refine the model's performance.

The importance of spam detection cannot be overstated, as it plays a crucial role in maintaining the safety and trust of mobile communication users. This project contributes to the ongoing efforts to combat spam and protect users from malicious activities.

---

7. Challenge and solution

a.Class Imbalance

- **Issue**: Spam messages are much less frequent compared to ham messages, resulting in a class imbalance where the model may be biased toward predicting the majority class (ham). This can lead to lower performance in detecting the less

frequent class (spam).

- **Impact**: The model might achieve high accuracy overall but perform poorly in identifying spam messages.
- **Solution**: Address this imbalance using techniques such as:
    - **Resampling**: Over-sample the minority class (spam) or under-sample the majority class (ham).
    - **Class Weight Adjustment**: Modify the class weights in the loss function to give more importance to the minority class.
    - **Synthetic Data Generation**: Use techniques like SMOTE to generate synthetic samples for the minority class.

b. Overfitting

- **Issue**: The model demonstrated high accuracy on the training data but did not generalize as well on the test data, indicating overfitting. This means the model has learned the details and noise in the training data rather than generalizing from it.
- **Impact**: Overfitting leads to a model that performs well on training data but poorly on unseen data, reducing its effectiveness in real-world scenarios.
- **Solution**: Mitigate overfitting by:
    - **Regularization**: Apply techniques like L1 or L2 regularization to penalize large weights.
    - **Dropout**: Introduce dropout layers to randomly drop units during training, reducing reliance on specific neurons.
    - **Early Stopping**: Monitor validation performance and stop training when performance starts to degrade.
    - **Data Augmentation**: Increase the amount of training data through various augmentation techniques to improve generalization.

---

8. References

- Spam Email Statistics: An Overview of the Growing Threat of Spam (Yearly Data)
- TensorFlow and Keras Documentation for Deep Learning Model Development
- Research Papers and Projects on SMS Spam Detection and Natural Language Processing

---

This report provides a comprehensive overview of the project, highlighting the motivation, objectives, methodology, results, and future directions for improving SMS spam detection.