SMS Spam Detection

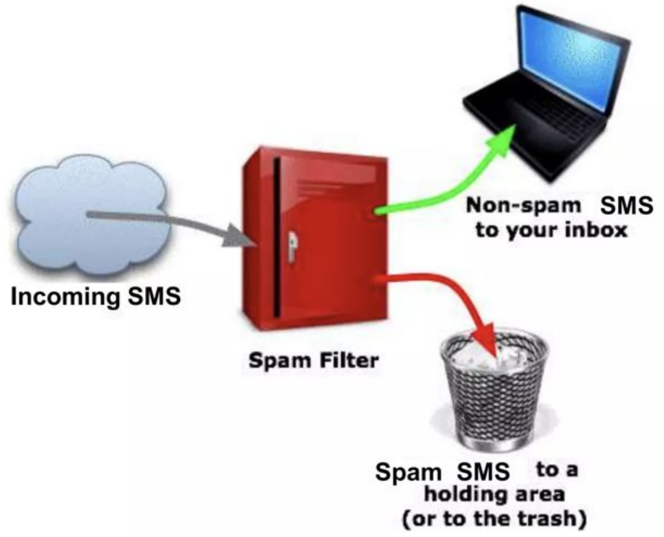
Using NLP and Deep Learning Methods

Group 4
Emily Weng - 20016
Huijuan Xie-19893

Content

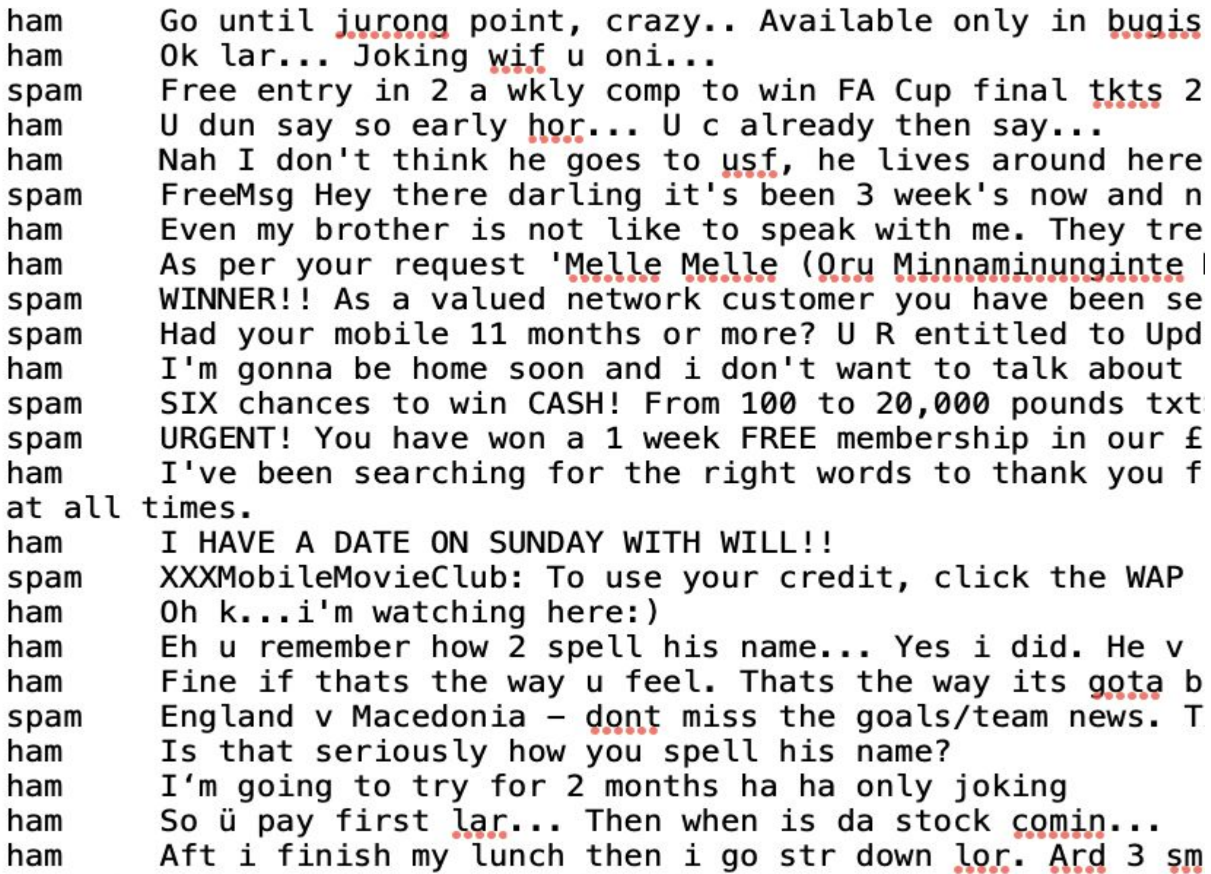
Introduction

- widespread use of mobile devices has caused a significant rise in spam messages.
- often used by hackers for financial gain or phishing, pose serious threats to user safety and trust.
- developing a deep learning model to accurately differentiate between spam and legitimate SMS messages, contributing to safer and more secure communication.


Incoming SMS
Spam Filter
Non-spam SMS to your inbox
Spam SMS to a holding area (or to the trash)

Dataset Overview

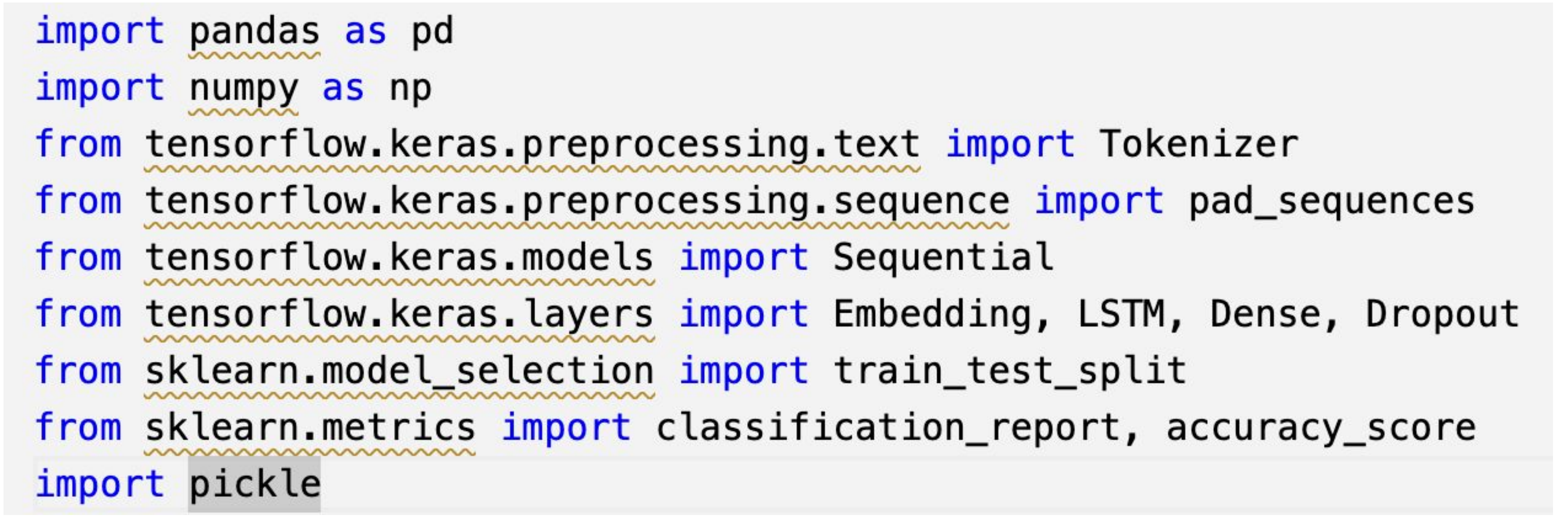
- We will use a pre-made dataset from [UCI SMS Spam Dataset]



```
ham     Go until jurong point, crazy.. Available only in bugis
ham     Ok lar... Joking wif u oni...
spam    Free entry in 2 a wkly comp to win FA Cup final tkts 2
ham     U dun say so early hor... U c already then say...
ham     Nah I don't think he goes to usf, he lives around here
spam    FreeMsg Hey there darling it's been 3 week's now and n
ham     Even my brother is not like to speak with me. They tre
ham     As per your request 'Melle Melle (Oru Minnaminunginte
spam    WINNER!! As a valued network customer you have been se
spam    Had your mobile 11 months or more? U R entitled to Upd
ham     I'm gonna be home soon and i don't want to talk about
spam    SIX chances to win CASH! From 100 to 20,000 pounds txt
spam    URGENT! You have won a 1 week FREE membership in our £
ham     I've been searching for the right words to thank you f
at all times.
ham     I HAVE A DATE ON SUNDAY WITH WILL!!
spam    XXXMobileMovieClub: To use your credit, click the WAP
ham     Oh k...i'm watching here:)
ham     Eh u remember how 2 spell his name... Yes i did. He v
ham     Fine if thats the way u feel. Thats the way its gota b
spam    England v Macedonia - dont miss the goals/team news. T
ham     Is that seriously how you spell his name?
ham     I'm going to try for 2 months ha ha only joking
ham     So ü pay first lar... Then when is da stock comin...
ham     Aft i finish my lunch then i go str down lor. Ard 3 sm
```

Data Preprocessing

We mainly use the keras library in the tensorflow framework for data processing, building deep learning models, and evaluating their performance, because Keras is easy to use and flexible in building neural networks.



```
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import pickle
```

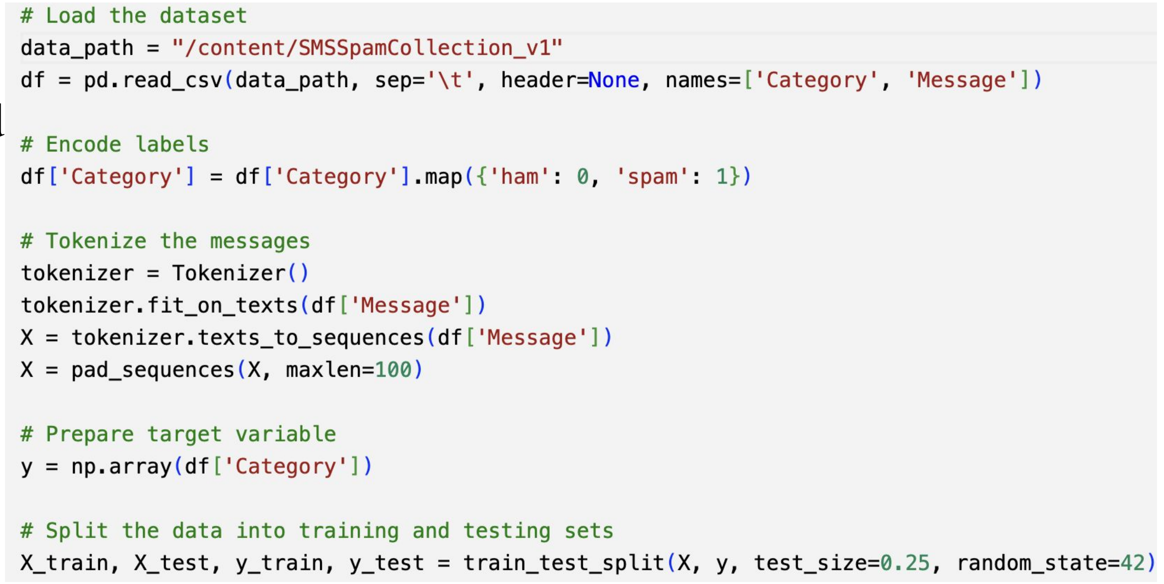
Data Preprocessing

Loading dataset: SMSSpamCollection_v1

Label Encoding: Spam is encoded as 1, and Ham is encoded as 0.

Tokenization: Use the Keras Tokenizer to convert text to sequences of integers.

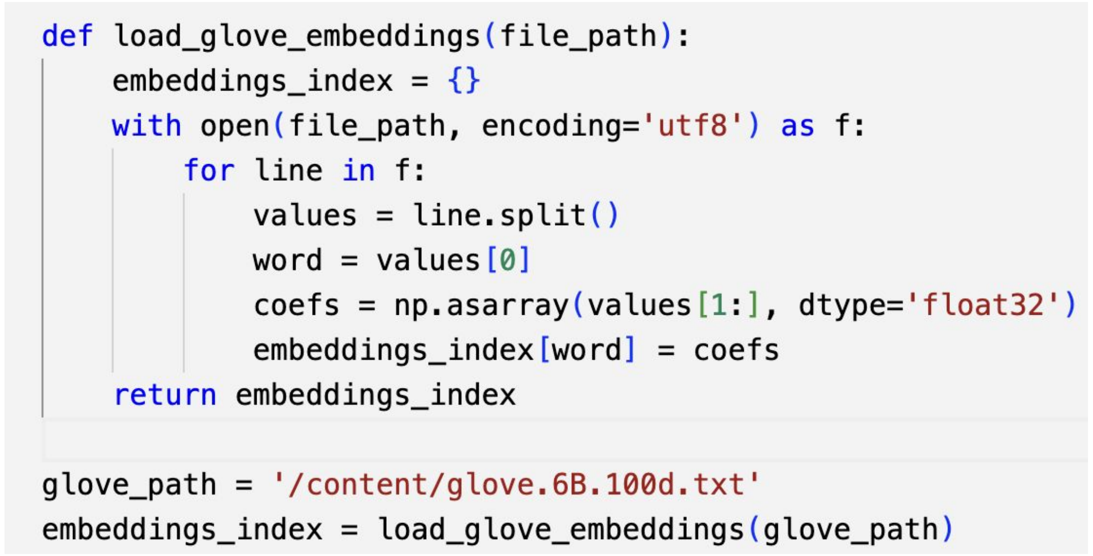
Padding: Padding sequences to a fixed length of 100 words.()



```
# 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)
```

Load GloVe embeddings

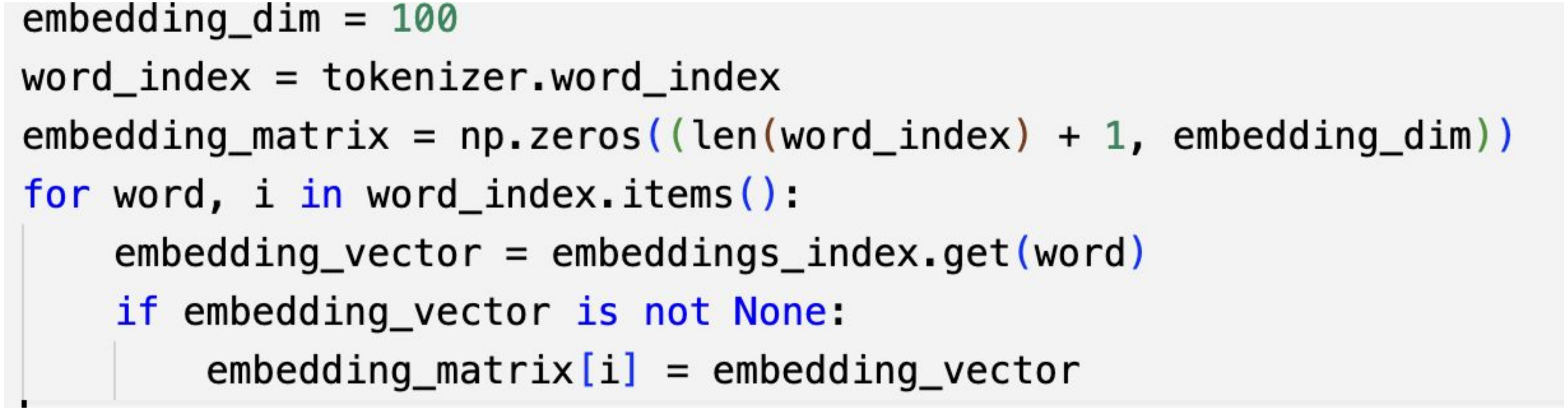
- This function loads pre-trained GloVe embeddings from a file and stores them in a dictionary, where each word is mapped to its corresponding embedding vector.
- It helps the model understand the semantic meaning of words without having to learn word representations from scratch, which can improve performance and reduce training time.



```python
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)
```

Prepare embedding matrix

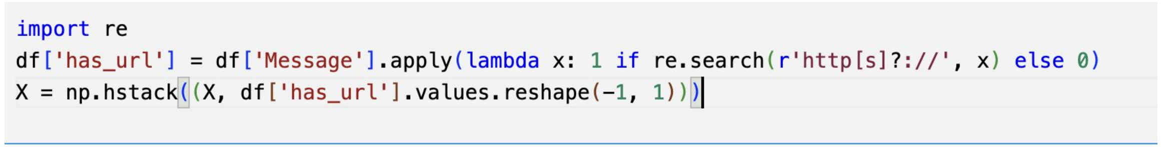
- The embedding matrix is created by mapping each word in the tokenizer's vocabulary to its corresponding GloVe embedding vector.
- The embedding matrix is used to initialize the embedding layer in the model, allowing it to start with meaningful word representations rather than random values.



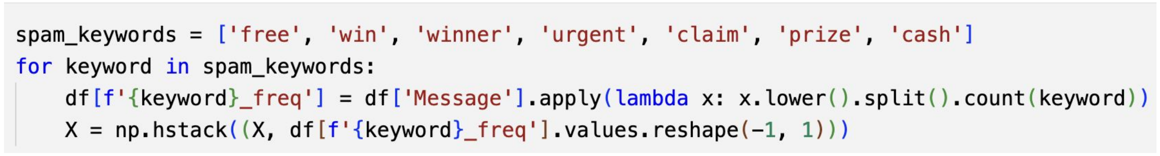
```
embedding_dim = 100
word_index = tokenizer.word_index
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Adding Additional Features

- The has_url feature indicates whether a message contains a URL.
- The frequency of specific spam-related keywords is calculated for each message.
- Adding these features helps the model better differentiate between ham and spam messages, as URLs and certain keywords are often indicative of spam.



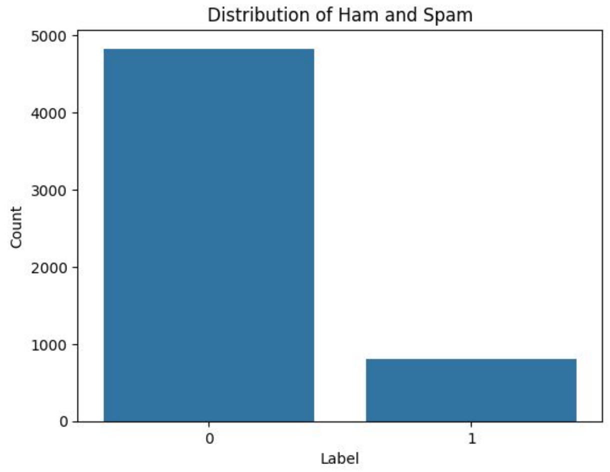
```python
import re
df['has_url'] = df['Message'].apply(lambda x: 1 if re.search(r'http[s]?://', x) else 0)
X = np.hstack((X, df['has_url'].values.reshape(-1, 1)))
```

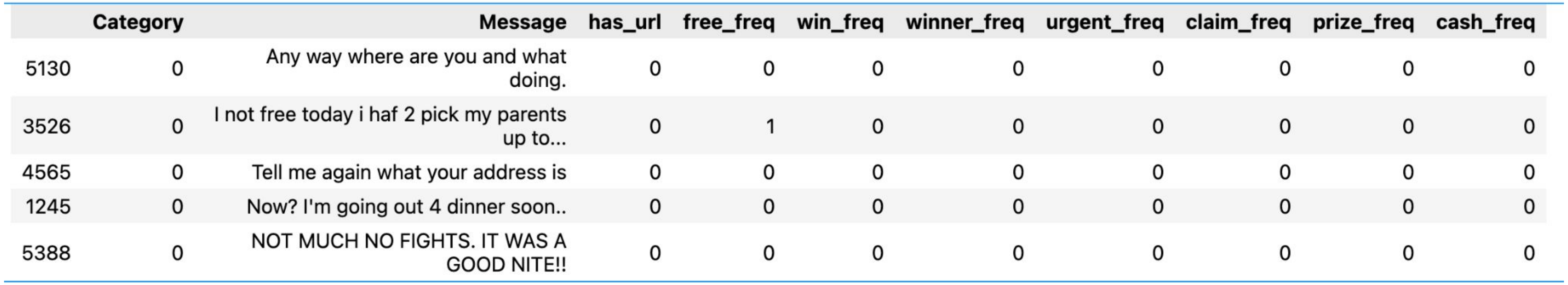


```python
spam_keywords = ['free', 'win', 'winner', 'urgent', 'claim', 'prize', 'cash']
for keyword in spam_keywords:
    df[f'{keyword}_freq'] = df['Message'].apply(lambda x: x.lower().split().count(keyword))
    X = np.hstack((X, df[f'{keyword}_freq'].values.reshape(-1, 1)))
```

Visualizing Data Distribution

- A count plot is generated to visualize the distribution of ham and spam messages
- After the above data processing, here are some samples of our data. It can help us build models better and more conveniently.



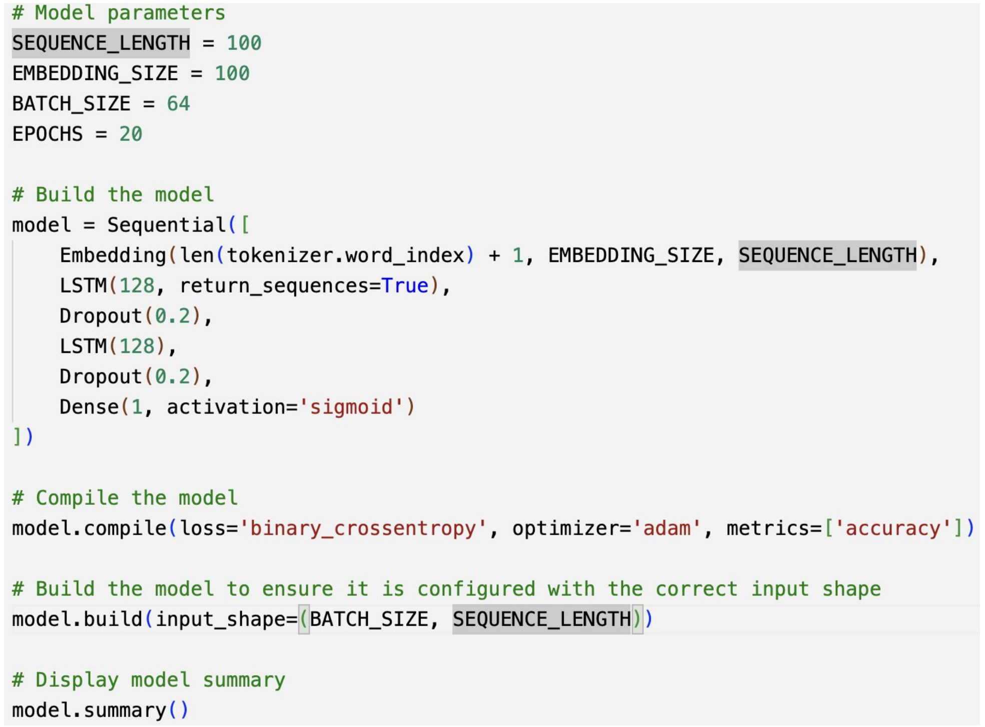


	Category	Message	has_url	free_freq	win_freq	winner_freq	urgent_freq	claim_freq	prize_freq	cash_freq
5130	0	Any way where are you and what doing.	0	0	0	0	0	0	0	0
3526	0	I not free today i haf 2 pick my parents up to...	0	1	0	0	0	0	0	0
4565	0	Tell me again what your address is	0	0	0	0	0	0	0	0
1245	0	Now? I'm going out 4 dinner soon..	0	0	0	0	0	0	0	0
5388	0	NOT MUCH NO FIGHTS. IT WAS A GOOD NITE!!	0	0	0	0	0	0	0	0

Model Architecture

Model Type: Sequential Model
Layers:

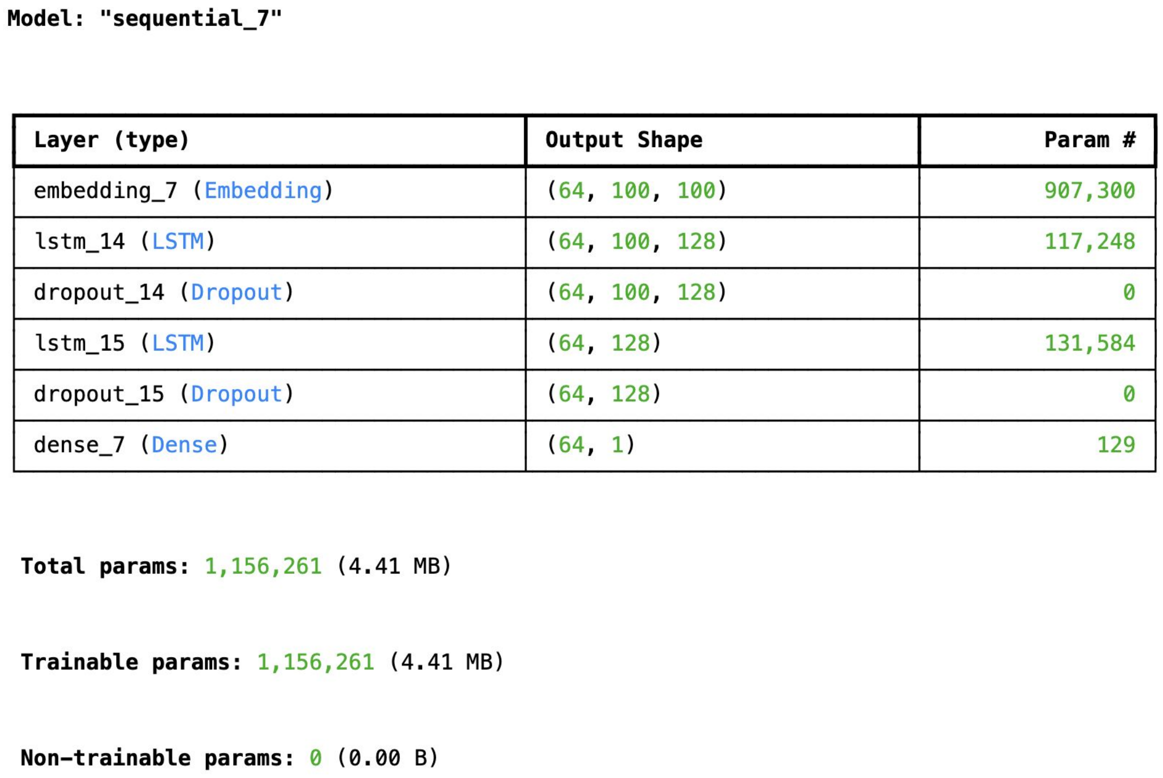
- Embedding Layer: Maps words to vectors.
- LSTM Layers: Capture temporal patterns in sequences.
- Dropout Layers: Prevent overfitting.
- Dense Layer: Output layer with sigmoid activation for binary classification.



```
# 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()
```

Model Architecture

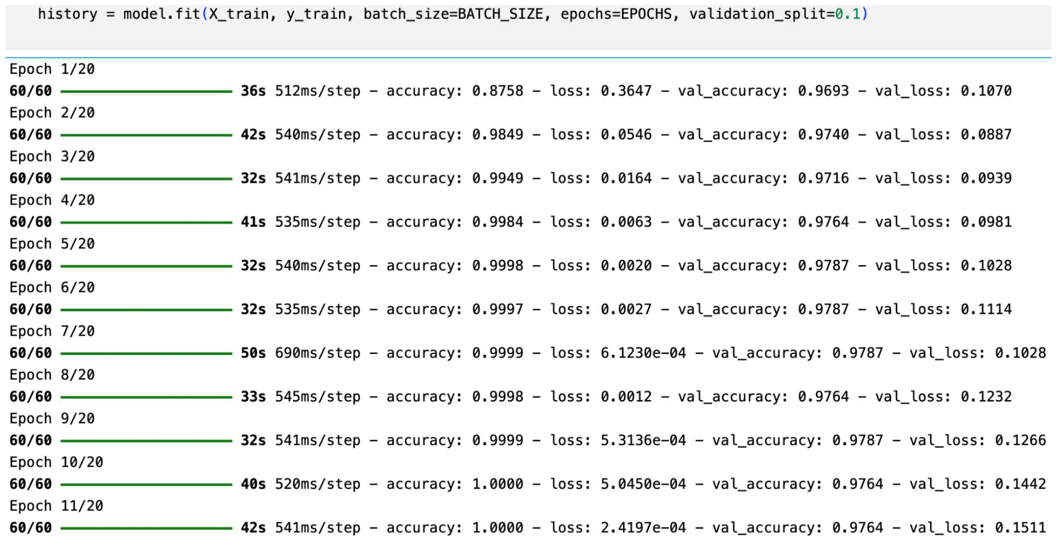
- This is a summary of the structure of our model, showing the type, output shape, and number of parameters of each layer.
- Total params: 1,156,261
- Trainable params: 1,156,261
- Non-trainable params: 0



```
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)
```

Model Training

- **Training Process:**
  - 75% of data used for training, with 10% of training data used for validation.
  - Loss function: Binary Cross Entropy
  - Optimizer: Adam
- **Accuracy Over Epochs:**
  - Graph showing training and validation accuracy.
- Using a validation split allows for monitoring the model's performance on unseen data during training, helping to prevent overfitting and determine when to stop training.

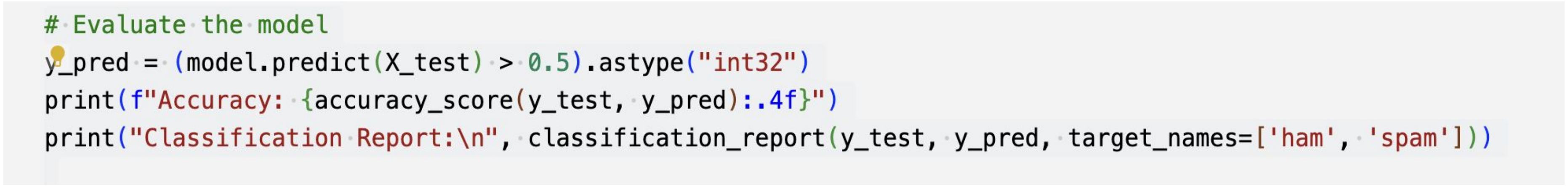


```
history = model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS, validation_split=0.1)
```

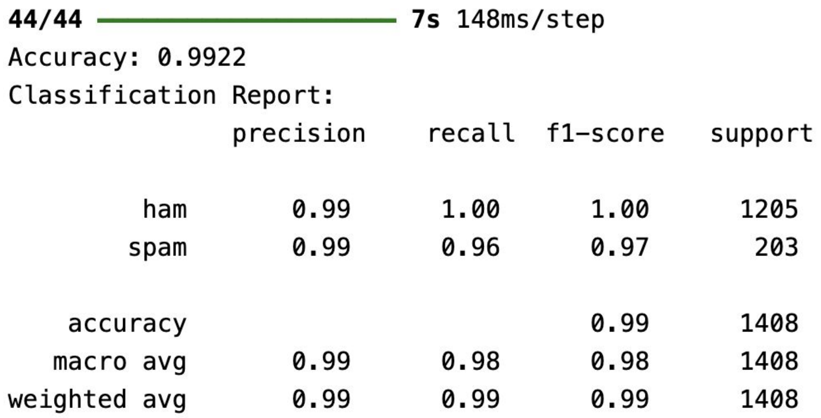
```
Epoch 1/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 36s 512ms/step - accuracy: 0.8758 - loss: 0.3647 - val_accuracy: 0.9693 - val_loss: 0.1070
Epoch 2/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 42s 540ms/step - accuracy: 0.9849 - loss: 0.0546 - val_accuracy: 0.9740 - val_loss: 0.0887
Epoch 3/20
60/60 ━━━━━━━━━━━━━━━━━━━━ 32s 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
```

Model Evaluation

- **Test Accuracy:**
  - Achieved accuracy: 99.22%



```python
# Evaluate the model
y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred, target_names=['ham', 'spam']))
```

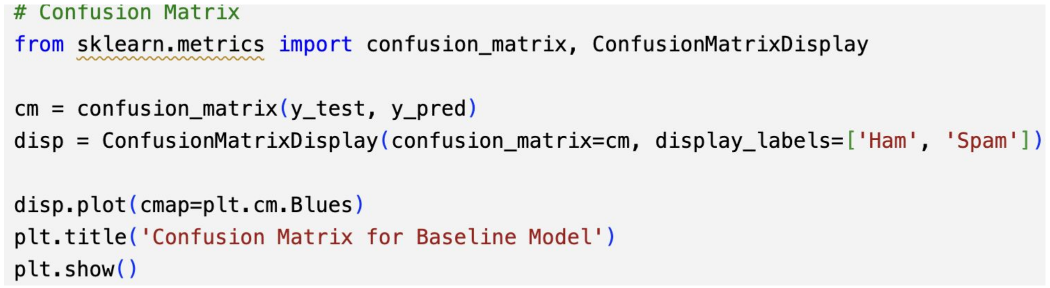


```
44/44 ━━━━━━━━━━━━━━━━━━━━ 7s 148ms/step
Accuracy: 0.9922
Classification Report:
               precision    recall  f1-score   support

         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
```

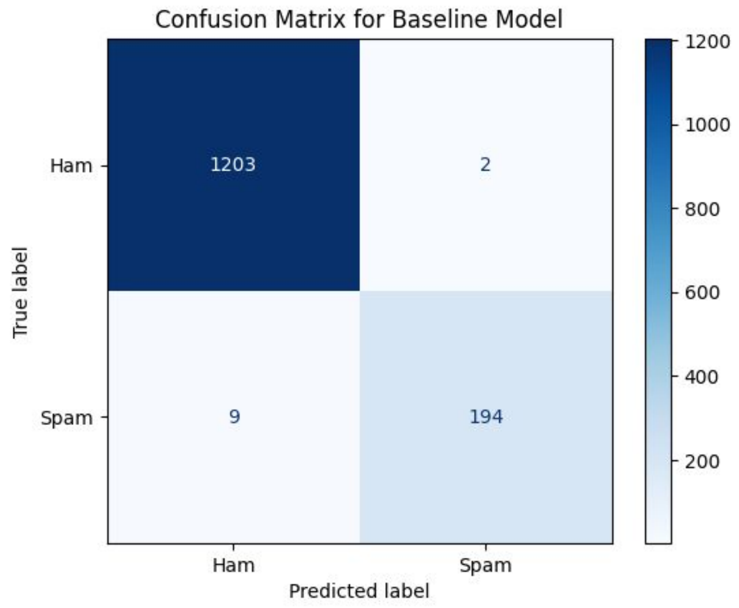
Model Evaluation

Confusion Matrix:

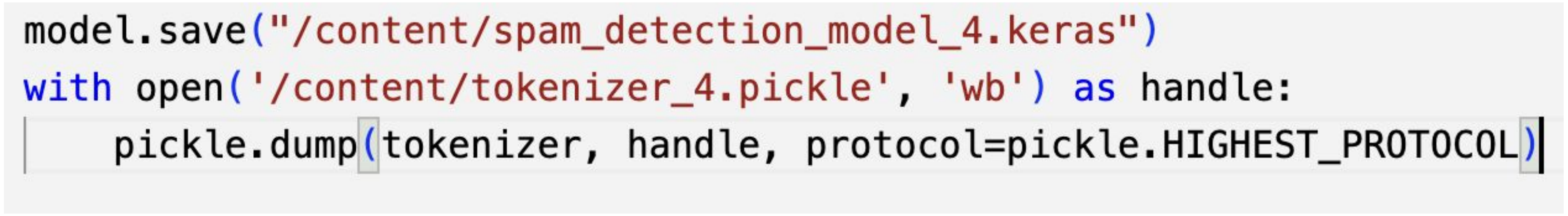
- Visual representation of True Positives, True Negatives, False Positives, and False Negatives.



```python
# Confusion Matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Ham', 'Spam'])

disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Baseline Model')
plt.show()
```



Model Evaluation

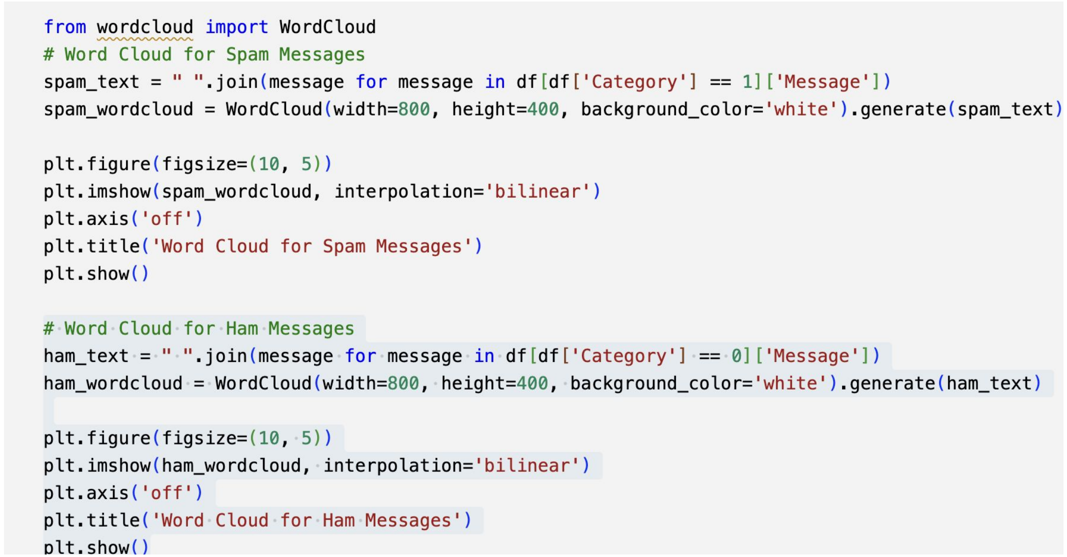


```
model.save("/content/spam_detection_model_4.keras")
with open('/content/tokenizer_4.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

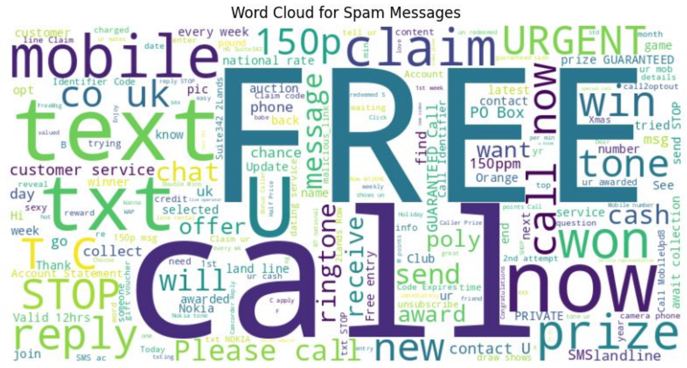
- The model is saved as a .keras file so that it can be reloaded when needed for prediction or further training.
- The tokenizer is saved as a .pickle file so that the text data can be processed in the same way in the future.
- Doing so can save time and computing resources because you don't need to retrain the model or reprocess the data every time.

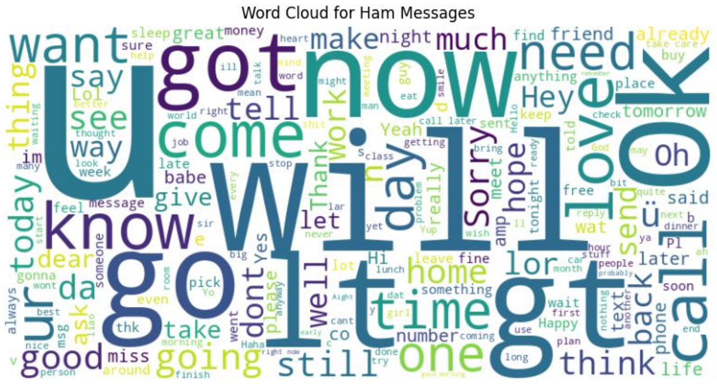
Word Cloud Analysis

- **Word Cloud for Spam Messages:**
  - Visual showing the most common words in spam messages.
- **Word Cloud for Ham Messages:**
  - Visual showing the most common words in ham messages.



```python
from wordcloud import WordCloud
# Word Cloud for Spam Messages
spam_text = " ".join(message for message in df[df['Category'] == 1]['Message'])
spam_wordcloud = WordCloud(width=800, height=400, background_color='white').generate(spam_text)

plt.figure(figsize=(10, 5))
plt.imshow(spam_wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Spam Messages')
plt.show()

# Word Cloud for Ham Messages
ham_text = " ".join(message for message in df[df['Category'] == 0]['Message'])
ham_wordcloud = WordCloud(width=800, height=400, background_color='white').generate(ham_text)

plt.figure(figsize=(10, 5))
plt.imshow(ham_wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Ham Messages')
plt.show()
```





Challenges and Limitations

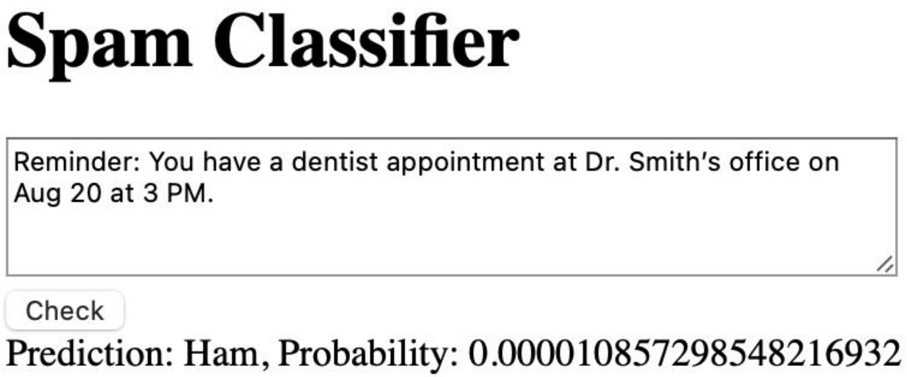
1.Class Imbalance

- **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.

2. 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:
  - **Dropout**: Introduce dropout layers to randomly drop units during training, reducing reliance on specific neurons.
  - **Model Complexity:** The architecture includes two LSTM layers, which are followed by dropout layers. By not making the model excessively deep or complex, it may help mitigate overfitting.

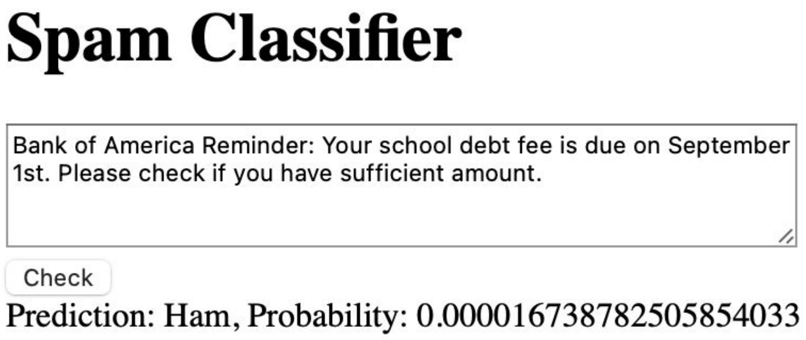
Examples:



Spam Classifier

Reminder: You have a dentist appointment at Dr. Smith's office on Aug 20 at 3 PM.

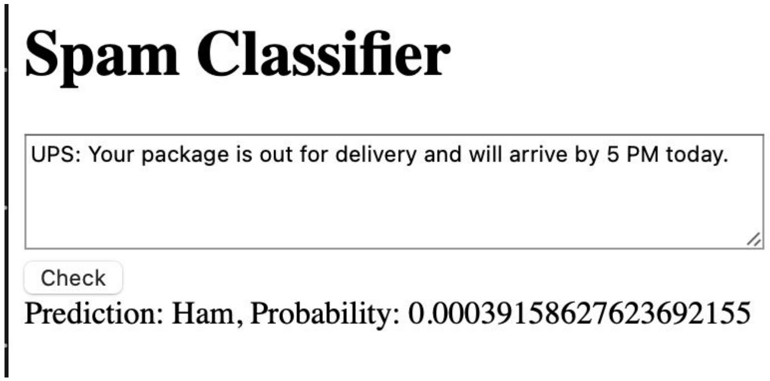
Prediction: Ham, Probability: 0.000010857298548216932



Spam Classifier

Bank of America Reminder: Your school debt fee is due on September 1st. Please check if you have sufficient amount.

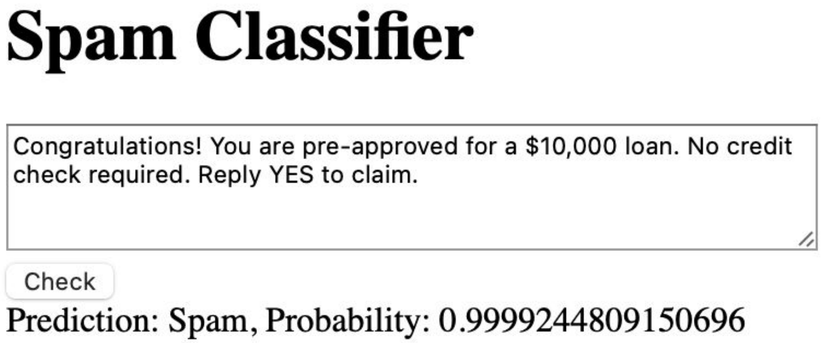
Prediction: Ham, Probability: 0.000016738782505854033



Spam Classifier

UPS: Your package is out for delivery and will arrive by 5 PM today.

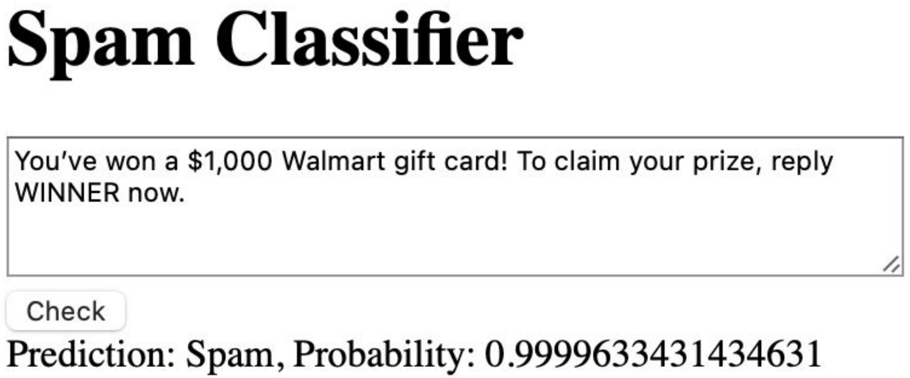
Prediction: Ham, Probability: 0.00039158627623692155



Spam Classifier

Congratulations! You are pre-approved for a $10,000 loan. No credit check required. Reply YES to claim.

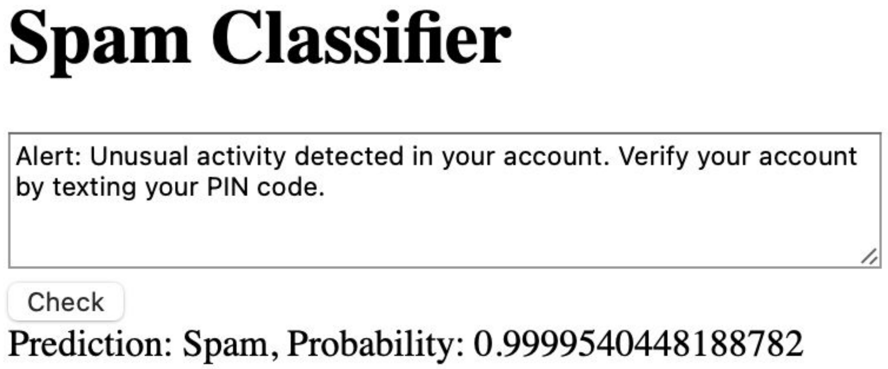
Prediction: Spam, Probability: 0.9999244809150696



Spam Classifier

You’ve won a $1,000 Walmart gift card! To claim your prize, reply WINNER now.

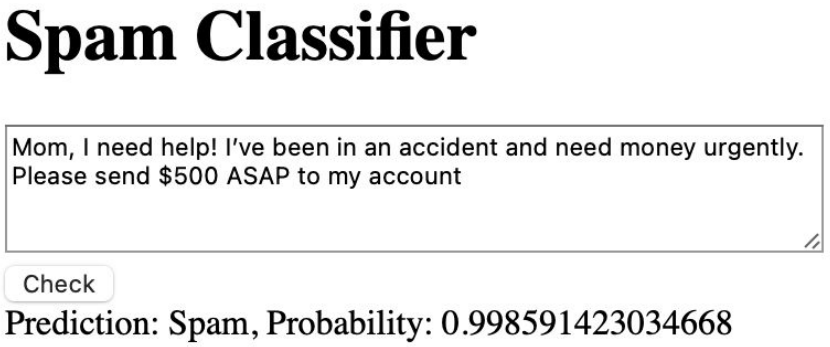
Prediction: Spam, Probability: 0.9999633431434631



Spam Classifier

Alert: Unusual activity detected in your account. Verify your account by texting your PIN code.

Prediction: Spam, Probability: 0.9999540448188782



Spam Classifier

Mom, I need help! I’ve been in an accident and need money urgently. Please send $500 ASAP to my account

Prediction: Spam, Probability: 0.998591423034668

Reference

1. UCI SMS Spam Dataset
2. What's On the Other Side of Your Inbox - 20 SPAM Statistics 2024
3. Gupta, S., & Garg, S. (2019). Machine Learning Models for SMS Spam Detection: A Review. International Journal of Engineering and Advanced Technology (IJEAT), 8(6), 4461-4467.
4. https://sist.sathyabama.ac.in/sist_naac/documents/1.3.4/1822-b.e-cse-batchno-109.pdf
    a. SMS SPAM DETECTION USING MACHINE LEARNING AND DEEP LEARNING TECHNIQUES
5. https://www.geeksforgeeks.org/sms-spam-detection-using-tensorflow-in-python/
    a. aravind sivaraj(2024), SMS Spam Detection using TensorFlow in Python

Thank you