SMS SPAM DETECTION

Pavani Billapati(Pbill1@unh.newhaven.edu)
Trilok kumar pidikiti(tpidi1@unh.newhaven.edu)
Harianth Kancherla(hkanc1@unh.newhaven.edu)

Abstract:

In the era of pervasive mobile communication, the surge in SMS spam presents a formidable threat, with malicious actors exploiting unsuspecting individuals through deceptive messages for extracting sensitive information. This research addresses the urgent need for effective SMS spam detection, aiming to curtail the risk of individuals falling victim to phishing attacks and incurring financial losses. We propose an innovative SMS spam detection system, leveraging advanced Natural Language Processing (NLP) techniques and employing Deep Learning architectures such as Long Short-Term Memory (LSTM), Bidirectional LSTM, and Dense Neural Networks.

Concentrating on English language SMS spams, our methodology encompasses meticulous data preparation involving word embedding, tokenization, padding, and truncation to enhance the dimensionality of the data. The model development phase incorporates a Dense Neural Network for baseline understanding and compares it with the sequence modeling capabilities of LSTM and the bidirectional nature of Bi-LSTM.

This research contributes a robust solution to SMS spam detection, emphasizing the efficacy of advanced NLP and Deep Learning techniques in fortifying individuals against phishing attempts and preserving their financial well-being. The diverse model architectures considered in our approach signify a comprehensive exploration of strategies to address the evolving landscape of SMS spam threats.

Introduction:

The state of communication technology is currently quickly advancing. This implies that with the use of email, text messaging, and web surfing, anyone may now obtain or receive information more easily than in the past. By sending a false notice of a bank transaction or a misleading advertisement, a hacker can use these technologies' benefits to obtain sensitive information from users, including phone numbers, credit card numbers, and online banking account information [1]. Furthermore, the unsuspecting individuals who received the message in a panic then obeyed the hacker's instructions, providing them with their private information. The hacker then utilizes this data to obtain assets from individuals.

Spam that is sent via mobile devices and targets users' sensitive information is known as short messaging service (SMS) spam [2], [3]. These days, a variety of people who follow the hacker's instructions and believe the information in the message are severely impacted by an SMS spam attack. Having a technology that can efficiently identify spam messages can help to mitigate this issue.

Deep Learning is currently a widely utilized data analysis technique since it often yields excellent prediction accuracy [4], [5]. Recurrent Neural Networks (RNNs) are the method that is commonly applied when assessing sequential data, like text data [5]. The LSTM, or long short-term memory, Bi-LSTM and Dense Model algorithms are adjusted for RNNs. Many published studies assert that LSTM Bi-LSTM and Dense

Model perform better in deep learning than other methods, especially for the analysis of sequential data. Taking Kraus and Feuerriegel as examples developed a decision support system to help investors feel more safe about their investments by using economic news and long short-term memory (LSTM). Moreover, they employ word tokenization, transfer learning, utilizing word embedding to get data ready.

In this study, we make use of Keras, a Python toolkit for developing deep learning models with a Tensorflow backend. Its data preparation toolset includes word tokenization, data padding and truncation, and word embedding. Using the word tokenization process, text inputs are converted into sequential data using the words' index values. While the word embedding approach is used to add extra dimensions to the sequence into the vector, To ensure that each sequence is the same length, the techniques of padding and truncating data are applied. After the data preparation step, we use the LSTM, Bi-LSTM and Dense Model algorithms to train the model. Next, we assess the models' performance.

Dataset:

We use an SMS spam dataset that was downloaded from UCI datasets. This collection has roughly 5,572 records. It includes English-language SMS text messaging chats with text and numbers in varying sentence lengths. Every record in this dataset has a label already. The label for the spam messages is ‘spam’ (747 records), while the label for the regular mails is ‘ham’(4,825 records).

```
[5]  messages.describe()
```

	label	message
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

Preparing Data:

Natural Language Process (NLP) is used in this procedure to pre-process natural language data. The goal of natural language processing (NLP) is to enable computers to comprehend natural language in the same way that humans do. It offers a plethora of methods for preprocessing data into a machine-readable format. In this study, we text input into sequential data, which we then use to build LSTM, Bidirectional LSTM, and Dense Model algorithms are used to create SMS classification models. For pre-processing data, we additionally employ word tokenization, padding, truncating, and word embedding algorithms. The following is a description of each method we employ in the data pre-processing step.

Word Tokenization:

The process of converting sentences words into index values represented by a number is called word tokenization. To develop a word tokenizer, we established a lot of intriguing vocabulary terms in this process. We utilize word token to convert the words in the sentence to serial data after generating the token. If the word is unknown, the tokenizer sets the index to 0 and converts the word to index. Furthermore, we

generate a tokenizer by assigning a fixed 10,000 vocabulary items. Additionally, we employ the tokenizer is used to convert text data into an index number word sequence.

```
word_index = tokenizer.word_index
word_index
```

```
{'<OOV>': 1,
 'to': 2,
 'you': 3,
 'a': 4,
 'i': 5,
 'call': 6,
 'the': 7,
 'your': 8,
 'u': 9,
 'for': 10,
 'is': 11,
 'and': 12,
 'free': 13,
 'now': 14,
 'have': 15,
 '2': 16,
 'or': 17,
 'in': 18,
 'on': 19,
 'ur': 20,
 'of': 21,
 '4': 22,
 'txt': 23,
 'are': 24,
 'from': 25,
```

Padding and Truncating Data:

Data Truncation and Padding During this procedure, we use the LSTM, BI-LSTM and Dense Model algorithms to ensure that every sequence in the dataset has the same length for training. Based on (1), we determine the optimal message length. Once the message's length has been optimized, we pad any data that is shorter than the ideal length by adding 0 to the start of the sequence until the data length corresponds to the ideal length. We start at the beginning and truncate the data until its length equals the ideal length. if the data's length exceeds that of the sample. Optimize is equal to mean (len(xi)) + 2×std(len(xi)).

where xi represents the dataset's records, mean(x) represents the mean of the data, std(x) represents the message length function. Based on (1), we calculate the ideal message length. function. Based on (1), we determine the optimal message length. After computation, the ideal length for cover data is 200, which accounts for 97.95% of datasets.

Consequently, we regulate padding and truncating in sequence using this ideal length.

Word Embedding:

The word embedding technique developed by Pennington et al. is employed in this study. Using this method, a preprocessed word sequence is transformed into a vector representation known as embedding space, which has It has more dimensions than traditional word data and is used to train LSTM, BI-LSTM, and Dense Model algorithms.. We utilize the word embedding approach to create additional dimensions for the data in sequence after padding and truncating it. We set the embedding size to 32.

Modeling

In this study, we are developing SMS spam classification models using the three deep learning algorithms listed below: Bidirectional LSTM, LSTM and Dense model. Detailed description of each algorithm below.

Long Short-Term Memory (LSTM):

Hochreiter and Schmidhuber, a German team, first introduced LSTM in 1997. To solve the vanishing problem, LSTM enhances the base RNN algorithm. LSTM incorporates cell states to remember or forget data. These cell states have a structure known as a cell gate. The cell gate is made up of four pieces. The input gate, the forget gate, the memory cell state gate, and the output gate.

The gate that controls the input data is known as the input. The forgotten gate is in charge of the previous hidden state. The previous hidden state is to be saved in the current hidden state's memory cell.

The memory cell state gate updates the data based on input and forget gate information. The output gate computes the network's output data based on memory-cell state.

```
#LSTM hyperparameters
n_lstm = 20
drop_lstm =0.2
# Define LSTM Spam detection architecture
lstm_model = Sequential()
lstm_model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
lstm_model.add(LSTM(n_lstm, dropout=drop_lstm))
lstm_model.add(Dense(1, activation='sigmoid'))
lstm_model.summary()

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

# Fitting the LSTM spam detection model
num_epochs = 30
early_stop = EarlyStopping(monitor='val_loss', patience=5)
history = lstm_model.fit(training_padded, train_labels, epochs=num_epochs, validation_data=(testing_padded, test_labels), callbacks=[early_stop], verbose=2)

# Save the model
lstm_model.save('LSTM_Spam_Detection.h5')
```

The architecture of the LSTM is very good at capturing sequence patterns and dependencies. This makes it ideal for natural language understanding (NLU) tasks. In the SMS spam detection context, it allows the model to understand the nuances of the spam messages and distinguish them from the legitimate ones.

By adjusting the hyperparameters (e.g., LSTM number of units), the model can capture long-lasting dependencies and improve its performance.

This code implements an LSTM-based neural network for SMS spam detection. Within the architecture, words are represented by an embedding layer, sequential patterns are captured by an LSTM layer, and binary classification is performed by a dense layer. The Adam optimizer and binary cross-entropy loss are used to train the model with early stopping, to prevent overfitting. The trained model is then saved for use later. Experimenting with model architecture and hyperparameter adjustments can help further optimize performance.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 50, 16)            8000

 lstm (LSTM)                 (None, 20)                2960

 dense_2 (Dense)             (None, 1)                 21

=================================================================
Total params: 10981 (42.89 KB)
Trainable params: 10981 (42.89 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/30
38/38 - 5s - loss: 0.6927 - accuracy: 0.5456 - val_loss: 0.6883 - val_accuracy: 0.5987 - 5s/epoch - 125ms/step
Epoch 2/30
38/38 - 1s - loss: 0.5276 - accuracy: 0.7339 - val_loss: 0.3041 - val_accuracy: 0.9030 - 841ms/epoch - 22ms/step
Epoch 3/30
38/38 - 1s - loss: 0.2526 - accuracy: 0.9238 - val_loss: 0.1973 - val_accuracy: 0.9465 - 833ms/epoch - 22ms/step
Epoch 4/30
38/38 - 1s - loss: 0.1584 - accuracy: 0.9598 - val_loss: 0.1788 - val_accuracy: 0.9498 - 855ms/epoch - 23ms/step
Epoch 5/30
38/38 - 1s - loss: 0.1333 - accuracy: 0.9674 - val_loss: 0.1798 - val_accuracy: 0.9498 - 834ms/epoch - 22ms/step
Epoch 6/30
38/38 - 1s - loss: 0.1123 - accuracy: 0.9699 - val_loss: 0.1825 - val_accuracy: 0.9465 - 868ms/epoch - 23ms/step
Epoch 7/30
38/38 - 1s - loss: 0.0984 - accuracy: 0.9757 - val_loss: 0.1555 - val_accuracy: 0.9599 - 823ms/epoch - 22ms/step
Epoch 8/30
38/38 - 1s - loss: 0.0688 - accuracy: 0.9841 - val_loss: 0.1638 - val_accuracy: 0.9532 - 842ms/epoch - 22ms/step
Epoch 9/30
38/38 - 1s - loss: 0.0787 - accuracy: 0.9791 - val_loss: 0.1611 - val_accuracy: 0.9532 - 832ms/epoch - 22ms/step
Epoch 10/30
38/38 - 1s - loss: 0.0580 - accuracy: 0.9858 - val_loss: 0.1625 - val_accuracy: 0.9599 - 837ms/epoch - 22ms/step
Epoch 11/30
38/38 - 1s - loss: 0.0448 - accuracy: 0.9900 - val_loss: 0.1689 - val_accuracy: 0.9565 - 935ms/epoch - 25ms/step
Epoch 12/30
38/38 - 1s - loss: 0.0322 - accuracy: 0.9941 - val_loss: 0.1673 - val_accuracy: 0.9532 - 1s/epoch - 35ms/step
```

BIDIRECTIONAL LSTM:

A variation of the classic Long Short-Term Memory (LSTM) neural network architecture, Bidirectional Long Short-Term Memory (Bi-LSTM) anticipated to extract contextual information from both previous and subsequent elements in a series. Information processing usually happens in a unidirectional fashion, starting at the beginning and ending at the end of a sequence, in many sequential data tasks. The model's

comprehension of the sequence's dependencies and context may be limited by this unidirectional method.

```
# Biderectional LSTM Spam detection architecture
bidirectional_lstm_model = Sequential()
bidirectional_lstm_model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
bidirectional_lstm_model.add(Bidirectional(LSTM(n_lstm, dropout=drop_lstm)))
bidirectional_lstm_model.add(Dense(1, activation='sigmoid'))
bidirectional_lstm_model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])
# Training
num_epochs = 30
early_stop = EarlyStopping(monitor='val_loss', patience=4)
history = bidirectional_lstm_model.fit(training_padded, train_labels, epochs=num_epochs,
                    validation_data=(testing_padded, test_labels),callbacks =[early_stop], verbose=2)
bidirectional_lstm_model.save('BiLSTM_Spam_Detection.h5')
```

```
Epoch 1/30
38/38 - 7s - loss: 0.6863 - accuracy: 0.6025 - val_loss: 0.6621 - val_accuracy: 0.8629 - 7s/epoch - 191ms/step
Epoch 2/30
38/38 - 1s - loss: 0.5354 - accuracy: 0.8360 - val_loss: 0.4017 - val_accuracy: 0.8863 - 1s/epoch - 35ms/step
Epoch 3/30
38/38 - 1s - loss: 0.3001 - accuracy: 0.9063 - val_loss: 0.1975 - val_accuracy: 0.9264 - 1s/epoch - 35ms/step
Epoch 4/30
38/38 - 1s - loss: 0.1194 - accuracy: 0.9640 - val_loss: 0.1671 - val_accuracy: 0.9365 - 1s/epoch - 33ms/step
Epoch 5/30
38/38 - 2s - loss: 0.0932 - accuracy: 0.9732 - val_loss: 0.1433 - val_accuracy: 0.9565 - 2s/epoch - 46ms/step
Epoch 6/30
38/38 - 2s - loss: 0.0735 - accuracy: 0.9749 - val_loss: 0.1536 - val_accuracy: 0.9498 - 2s/epoch - 62ms/step
Epoch 7/30
38/38 - 2s - loss: 0.0741 - accuracy: 0.9782 - val_loss: 0.1508 - val_accuracy: 0.9532 - 2s/epoch - 52ms/step
Epoch 8/30
38/38 - 1s - loss: 0.0406 - accuracy: 0.9908 - val_loss: 0.1526 - val_accuracy: 0.9565 - 1s/epoch - 33ms/step
Epoch 9/30
38/38 - 1s - loss: 0.0427 - accuracy: 0.9883 - val_loss: 0.1813 - val_accuracy: 0.9465 - 1s/epoch - 33ms/step
```

To overcome this constraint, Bi-LSTM combines two independent LSTM layers: one that processes the sequence forward (from the start) and the other that processes it backward (from the finish). The model can consider each element in the sequence's past and future context thanks to this bidirectional processing.

While the backward LSTM layer gathers data from the sequence's future elements, the forward LSTM layer gathers data from the sequence's past elements. The model obtains a more thorough grasp of the context and dependencies within the sequence by integrating the two directions.

Before the outputs from both LSTM layers are sent to the neural network's later layers, they are usually combined, frequently by concatenation. Information from both directions is combined in this concatenated representation.

Dense Model:

Every node (or neuron) in one layer of an artificial neural network is connected to every other layer's node through a dense model architecture, also known as a dense neural network or fully connected neural network. Put differently, there is no layer skip in the information flow, both forward and backward. The layers in this architecture are highly connected to one another.

```
# Define Dense model architecture
dense_model = Sequential()
dense_model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
dense_model.add(GlobalAveragePooling1D())
dense_model.add(Dense(n_dense, activation='relu'))
dense_model.add(Dropout(drop_value))
dense_model.add(Dense(1, activation='sigmoid'))
dense_model.summary()

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

# Fitting the dense spam detector model
num_epochs = 50
early_stop = EarlyStopping(monitor='val_loss', patience=3)
history = dense_model.fit(training_padded, train_labels, epochs=num_epochs, validation_data=(testing_padded, test_labels), callbacks=[early_stop], verbose=2)

# Save the model
dense_model.save('Dense_Spam_Detection.h5')
```

In a dense model, weighted connections and activation functions are used by later hidden layers to process the input layer's received data. The output layer generates the model's result. The dense connections between neurons in neighboring layers are what give rise to the word "dense." The weights corresponding to every connection and the biases of every neuron are among the

parameters of the model. During training, these weights and biases are modified using optimization algorithms such as gradient descent to minimize the discrepancy between the target values and the predicted output.

Regression, feature learning, and classification are just a few of the applications for which dense architectures are useful. On complicated tasks, though, they might become computationally costly and prone to overfitting, which would prompt the creation of more advanced architectures like convolutional neural networks (CNNs) and recurrent neural networks (RNNs) for uses.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 16)            8000

 global_average_pooling1d (  (None, 16)                0
 GlobalAveragePooling1D)

 dense (Dense)               (None, 24)                408

 dropout (Dropout)           (None, 24)                0

 dense_1 (Dense)             (None, 1)                 25

=================================================================
Total params: 8433 (32.94 KB)
Trainable params: 8433 (32.94 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/50
38/38 - 4s - loss: 0.6833 - accuracy: 0.5958 - val_loss: 0.6710 - val_accuracy: 0.6890 - 4s/epoch - 117ms/step
Epoch 2/50
38/38 - 0s - loss: 0.6497 - accuracy: 0.7481 - val_loss: 0.6211 - val_accuracy: 0.8328 - 403ms/epoch - 11ms/step
Epoch 3/50
38/38 - 0s - loss: 0.5844 - accuracy: 0.8510 - val_loss: 0.5446 - val_accuracy: 0.8562 - 331ms/epoch - 9ms/step
Epoch 4/50
38/38 - 0s - loss: 0.5010 - accuracy: 0.8711 - val_loss: 0.4588 - val_accuracy: 0.8896 - 224ms/epoch - 6ms/step
Epoch 5/50
38/38 - 0s - loss: 0.4189 - accuracy: 0.8837 - val_loss: 0.3820 - val_accuracy: 0.9064 - 320ms/epoch - 8ms/step
Epoch 6/50
38/38 - 0s - loss: 0.3556 - accuracy: 0.8996 - val_loss: 0.3193 - val_accuracy: 0.9197 - 259ms/epoch - 7ms/step
Epoch 7/50
38/38 - 0s - loss: 0.2988 - accuracy: 0.9155 - val_loss: 0.2728 - val_accuracy: 0.9264 - 217ms/epoch - 6ms/step
Epoch 8/50
38/38 - 0s - loss: 0.2491 - accuracy: 0.9272 - val_loss: 0.2385 - val_accuracy: 0.9365 - 219ms/epoch - 6ms/step
```

An embedding layer for word mapping and a global average pooling 1D layer for reducing spatial dimensions define the Dense model for SMS spam detection. There are two dense layers: a dropout layer for regularization and a first layer with 24 units and ReLU activation. Binary predictions are generated by the final Dense layer, which has a sigmoid activation. The accuracy metric, Adam optimizer, and binary cross-entropy loss are used to compile the model. Training lasts 50 epochs, with an early termination occurring after 3 epochs in which the validation loss does not improve. The trained model is stored for later use, and the model successfully detects spam. Optimization can be achieved by experimenting with model architecture and hyperparameter adjustments.

Evaluation:

For loss and accuracy, all Dense, LSTM, and Bi-LSTM models are similar. These three models have validation losses of 0.14, 0.16, and 0.18, respectively. Furthermore, the respective validation accuracy rates are 95.99%, 95.32%, and 94%.

As a final model, we decide to classify the text messages as spam or ham using Dense architecture. Compared to LSTM and Bi-LSTM, the accuracy and loss over epochs are more stable in the dense classifier's straightforward structure.

```
# Comparing three different models
print(f"Dense architecture loss and accuracy: {dense_model.evaluate(testing_padded, test_labels)} " )
print(f"LSTM architecture loss and accuracy: {lstm_model.evaluate(testing_padded, test_labels)} " )
print(f"Bi-LSTM architecture loss and accuracy: {bidirectional_lstm_model.evaluate(testing_padded, test_labels)} " )
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.1424 - accuracy: 0.9599
Dense architecture loss and accuracy: [0.1424040049314499, 0.9598662257194519]
10/10 [==============================] - 0s 6ms/step - loss: 0.1673 - accuracy: 0.9532
LSTM architecture loss and accuracy: [0.16733580827713013, 0.953177273273468]
10/10 [==============================] - 0s 9ms/step - loss: 0.1813 - accuracy: 0.9465
Bi-LSTM architecture loss and accuracy: [0.1813274770975113, 0.9464883208274841]
```

Predictions:

First scenario: Using our data's raw text.

The third message below is spam, while the first and second are ham. To turn them into sequences, we employed the same tokenizer that we had previously written in the code. By doing this, the training sets token and the new words' token are guaranteed to match. After tokenization, we apply padding in the same way as previously and supply the same dimension as the training set.

```
# display long string
pd.options.display.max_colwidth=100
messages[:3]
```

	label	message
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive ...

The model correctly predicts that the third sentence is spam, while the first two sentences are not. The third sentence has a 99% probability of being spam..

```
# Defining prediction function
def predict_spam(predict_msg):
    new_seq = tokenizer.texts_to_sequences(predict_msg)
    padded = pad_sequences(new_seq, maxlen =max_len,
                           padding = padding_type,
                           truncating=trunc_type)
    return (dense_model.predict(padded))
predict_spam(predict_msg)
```

```
1/1 [==============================] - 0s 139ms/step
array([[0.01186293],
       [0.02574334],
       [0.9989698 ]], dtype=float32)
```

Scenario 2: Use newly created text messages to see how the model classifies them.

The first sentence below is more like spam, while the remaining two sentences are more like ham.

```
# The third one gives 0.99 indicating very high probabilty of spam
predict_msg = ["You are awarded a Nikon Digital Camera. Call now",
               "Call me",
          "What's up?"]
predict_spam(predict_msg)
```

```
1/1 [==============================] - 0s 29ms/step
array([[0.7814135 ],
       [0.03495146],
       [0.01337497]], dtype=float32)
```

According to our model, the first message has a 78% probability of being spam, while the remaining messages are classified as ham.

REFERENCES:

1. "Review spam detection," N. Jindal and B. Liu, Proceedings of the 16th International Conference on World Wide Web, 2007, pp. 1189–1190.

2. "Towards SMS spam filtering: Results under a new dataset," T. A. Almeida, J. M. G. Hidalgo, and T. P. Silva, International Journal of Information Security Science, vol. 2, No. 1, pp. 1-18, 2013.

3. "Text normalization and semantic indexing to enhance instant messaging and SMS spam filtering," Knowledge-Based Systems, vol. 108, pp. 25–32, 2016, T. A. Almeida, T. P. Silva, I. Santos, and J. M. G. Hidalgo.

4. J. Schmidhuber, "An overview of deep learning in neural networks," Neural Networks, vol. 61, 2015, pp. 85–117.

5. "Deep learning," Y. LeCun, Y. Bengio, and G. Hinton, Nature, vol. 521, no. 7553, pp. 436-444,