សកលវិទ្យាល័យភូមិន្ទភ្នំពេញ

ROYAL UNIVERSITY OF PHNOM PENH

ប្រធានបទរបាយការណ៍ស្រាវជ្រាវ

Fake News Classification Using LSTM Model

A Research Report
In Partial Fulfilment of the Requirement for Year 3
Project Practicum for Degree of Bachelor in Data Science and Engineering

Tham Veasna

July 2024

TABLE OF CONTENTS

List of Tables

List of Figures

List of Abbreviation

NLP	Natural Language Processing
LSTM	Long Short-Term Memory
NLTK	Natural Language Toolkit
TF-IDF	Term Frequency – Inverse Document Frequency
CNN	Convolutional Neural Network
BERT	Bidirectional Encoder Representations from Transformers
Bi-LSTM	Bidirectional Long Short-Term Memory
KNN	K – Nearest Neighbor

CHAPTER 1
INTRODUCTION

1.1 Background to the Study

The idea of spreading false information is not a recent development. It has happened since long before the advent of newspaper and digital age, misinformation spread through more traditional means. Gossiping, a prevalent form of communication in small communities, allowed rumors and unverified stories to circulate rapidly. These oral transmissions often embellished or distorted facts, creating a fertile ground for misinformation.

In addition to verbal gossip, the written word also played a significant role in dissemination of false information. Letters, a primary means of communication before the mass production of newspapers, often carried unverified rumors and hearsay. These letters, exchanged between individuals and communities, could spread misinformation over vast distances, influencing public opinion and social behavior.

In the past few centuries, with the emergence of newspaper, the scale and speed of spreading false information increased dramatically. And even more so in modern era, where the internet and social media have exponentially amplified the speed and reach of false information. Thus, the term “fake news” is used to describe false or misleading information presented as news. Unlike gossiping and letters, fake news today can reach millions of people within minutes, making it a significant concern for society.

1.2 Aim and Objective

The idea of this study is to develop, train and implement a fake news detection system that leverages machine learning and natural language processing techniques to accurately identify and mitigate the spread of false information into a website for users to identify fake news.

1.3 Limitation and Scope

This study aims to classify news into real or fake based on the WELFake dataset, which contains 72,134 data. According to IEEE, the publisher on WELFake research paper. There is supposed to be 35,028 real news and 37,106 fake news. This dataset focuses on the US political news and are written in English. From exploring this dataset, there seems to be a few limitations. Due to this dataset comes from the combination of 4

datasets (Kaggle, McIntire, Reuters and Buzzfeed Political), some news are the same, therefore this dataset having about 9,012 duplicate news especially on fake news which then create an imbalance dataset that consists of 34,791 real news and 28,331 fake news which may make the model leans more toward predicting real news than fake news.

1.4 Structure of Study

This study has 6 chapters and are organized as follows:

Chapter 1: Background knowledge, objective and limitation on fake news classification.

Chapter 2: provides a brief review of past studies and state of art

Chapter 3: addresses the methodology of the study

Chapter 4: explore the implementation of techniques

Chapter 5: show the result from training and testing and integration into website

Chapter 6: conclusion and future work

CHAPTER 2
LITURATURE REVIEW

2.1 Past Studies

In recent period of time, there are numerous researchers who have proposed and have done research and train fake news classification model by using various machine learning, deep learning and NLP technologies in order to identify and help in combatting against the spread of fake news. To gain better understanding of this type of text classification, the details of some research articles are summarized below:

Fake News Detection on Social Media: A Data Mining Perspective by Michigan State University have stated the basic and well-explained definition, problem and the flow of developing a fake news classification model. They also mention good website for datasets, formulas for evaluation metrics and related areas where the same model can be applied to.

Machine Learning Algorithms via Detection of Fake News by Manav Rachna International Institute of Research & Studies (MRIIRS) has stated that they have used 14 classifiers and 9 models of stacking classifiers. Their model 6 and 7 achieved the best accuracy of 96.13% while having large computational complexity. They have utilized NLTK library for data preprocessing, and 3 vectorizers (CountVectorizer, TF-IDF, HashingVectorizer) and 3 classifiers (Functional classifiers, Tree based classifiers and Probability based classifier), Logistic Regression, Arbitrary Timberland Classifiers an Multilayer Perceptron. From their result, TF-IDF and HashingVectorizer improved the calculation time with Multilayer Perceptron as their best and precise classifier to use among those that they have tried.

IFND: a benchmark dataset for fake news detection have stated that they would use both machine learning and deep learning classifiers for text classification and augmentation techniques to increase the accuracy of their models. The results are Naïve Bayes 87.5%, KNN, 90.2%, Decision Tree, 91.4%, Logistic Regression 93.3%, Random Forest 94%, LSTM 92.6% and Bi-LSTM 92.7%.

2.2 State of Art

2.2.1 NLTK

Natural Language Toolkit (NLTK) is one of the largest Python libraries for performing various Natural Language Processing tasks. From rudimentary tasks such as text pre-processing to tasks like vectorized representation of text. It can perform a variety of operations on textual data, such as classification, tokenization, stemming, tagging, parsing, semantic reasoning, lemmatization, etc.

2.2.1.1 WordNet

WordNet is a lexical database for the English language, grouping words into sets of synonyms called synsets, providing short definitions and usage examples, and recording various semantic relations between these synonym sets. It is an essential resource for many natural language processing tasks. NLTK provides a comprehensive interface to WordNet, allowing users to access synsets, hypernyms, hyponyms, meronyms, and other lexical relations. Here are some key functionalities:

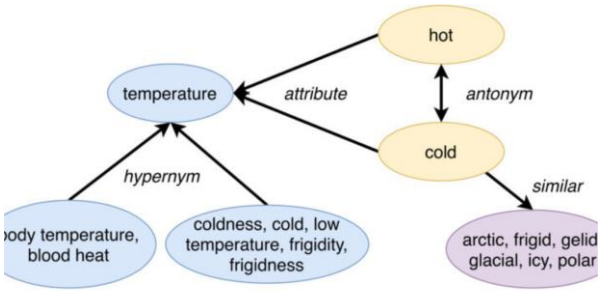

Figure 1: WordNet Lexical Relations

1. **Synsets:** Retrieve sets of synonyms for a given word.
2. **Hypernyms and Hyponyms:** Access broader (hypernyms) or more specific (hyponyms) terms.
3. **Definitions and Examples:** Obtain definitions and example sentences for synsets.
4. **Semantic Similarity:** Calculate the semantic similarity between different synsets.

2.2.1.2 Tokenization and Preprocessing

Tokenization is the process of splitting text into smaller units called tokens, which can be words, sentences, or subwords. Preprocessing also includes steps like stemming and lemmatization, part-of-speech tagging, parsing, and semantic reasoning to prepare text data for analysis.

Stemming is used for reducing words to their base or root form. NLTK provides several stemmers, including the Porter and Lancaster stemmers. Whereas, Lemmatization is similar to stemming but more accurate, as it considers the context and coverts words to

their meaningful base form. Part-of-Speech (POS) tagging helps in understanding the grammatical structure of the text. Parsing involves analyzing the grammatical structure of sentences. NLTK provides context-free grammars (CFG) for syntactic parsing. Semantic reasoning involves the understanding of the meaning and relationships between words in a text. These tools are the fundamental for preparing text data for further analysis or modeling in NLP tasks, making NLTK a powerful toolkit for NLP research and development.

2.2.2 TF-IDF

TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus (data-set).

Terminologies:

- **Term Frequency:** In document d, the frequency represents the number of instances of a given word t. Therefore, we can see that it becomes more relevant when a word appears in the text, which is rational. Since the ordering of terms is not significant, we can use a vector to describe the text in the bag of term models. For each specific term in the paper, there is an entry with the value being the term frequency.

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF-IDF(t,d) = TF(t,d) * IDF(t)$$

Figure 2: TF-IDF calculation formulas

- **Document Frequency:** This tests the meaning of the text, which is very similar to TF, in the whole corpus collection. The only difference is that in document d, TF is the frequency counter for a term t, while df is the number of occurrences in the document set N of the term t. In other words, the number of papers in which the word is present is DF.
- **Inverse Document Frequency:** Mainly, it tests how relevant the word is. The key aim of the search is to locate the appropriate records that fit the demand. Since tf considers all terms equally significant, it is therefore not only possible to use the term frequencies to measure the weight of the term in the paper.

- **Computation:** TF-IDF is one of the best metrics to determine how significant a term is to a text in a series or a corpus. TF-IDF is a weighting system that assigns a weight to each word in a document based on its term frequency (tf) and the reciprocal document frequency (TF) (IDF). The words with higher scores of weights are deemed to be more significant.

Usually, the TF-IDF weight consists of two terms:

1. Normalized Term Frequency (TF)
2. Inverse Document Frequency (IDF)

2.2.3 LSTM

Long Short-Term Memory (LSTM) is an improved version of recurrent neural network (RNN) designed by Hochreiter and Schmidhuber. A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTM model address this problem by introducing a memory cell, which is a container that can hold information for an extended period.

The LSTM architectures involves the memory cell which is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell.

- The input gate controls what information is added to the memory cell.
- The forget gate controls what information is removed from the memory cell.
- The output gate controls what information is output from the memory cell.

This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.

The LSTM maintains a hidden state, which acts as the short-term memory of the network. The hidden state is updated based on the input, the previous hidden state, and the memory cell's current state.

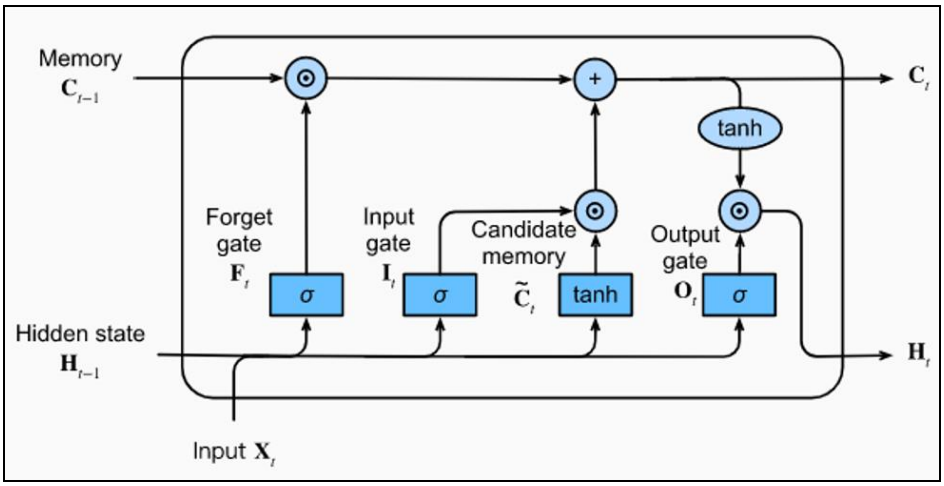

Figure 3: LSTM Model Architecture

2.2.3.1 Bi - LSTM

Bidirectional LSTM (Bi LSTM/ BLSTM) is recurrent neural network (RNN) that is able to process sequential data in both forward and backward directions. This allows Bi LSTM to learn longer-range dependencies in sequential data than traditional LSTMs, which can only process sequential data in one direction.

- Bi LSTMs are made up of two LSTM networks, one that processes the input sequence in the forward direction and one that processes the input sequence in the backward direction.
- The outputs of the two LSTM networks are then combined to produce the final output.

2.2.3.2 LSTM Working

LSTM architecture has a chain structure that contains four neural networks and different memory blocks called cells. Information is retained by the cells and the memory manipulations are done by the **gates.** There are three gates:

+ Forget Gate:

The forget gate in an LSTM (Long Short-Term Memory) network decides which information to discard from the cell state. It takes two inputs: (the current input) and (the previous hidden state). These inputs are multiplied by weight matrices and added to a

bias. The resultant value is then passed through a sigmoid activation function, which outputs a value between 0 and 1. An output of 0 means the information is forgotten, while an output of 1 means the information is retained. The forget gate equation is:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

+ Input Gate:

The input gate in an LSTM network is responsible for adding useful information to the cell state. It operates as follows:

1. **Regulate Information:** The (current input) and (previous hidden state) are processed through a sigmoid function, determining which values to remember. This produces:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

2. **Create Candidate Vector:** A vector of candidate values is generated using the tanh function, which outputs values between -1 and +1:

$$\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

3. **Update Cell State:** The previous cell state is multiplied by the forget gate output to discard certain information. Then, the regulated values are multiplied by the candidate vector to add new information to the cell state:

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

+ Output Gate

The output gate in an LSTM extracts useful information from the cell state by first applying the tanh function to create a vector. This vector is then regulated using the

sigmoid function. The resulting values are multiplied to produce the output and the input for the next cell. The equation is:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

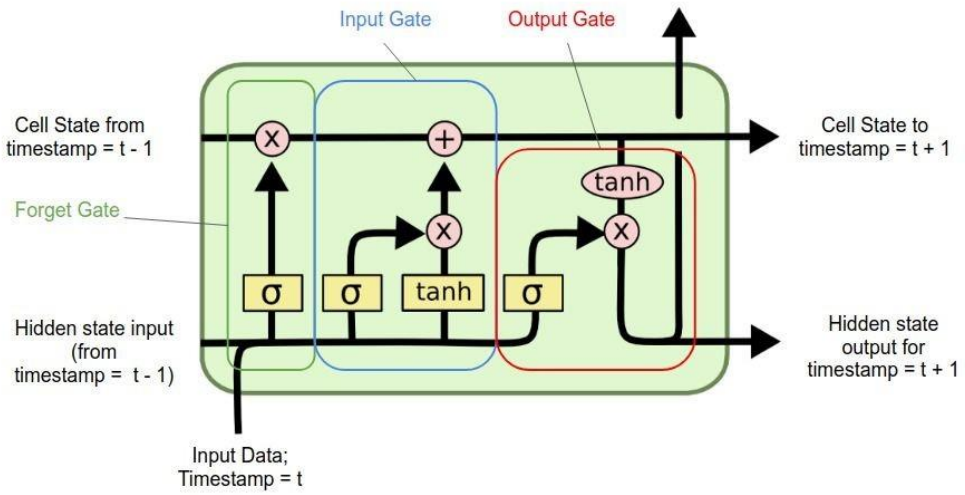

Figure 4: LSTM Gates Architecture

CHAPTER 3
METHODOLOGY

3.1 Method Overview

This section of the study provides the architecture of the model. It also contains details related to the datasets which is used to train an evaluate this model. A brief background dataset was used, and preprocessing methods are explained in this section.

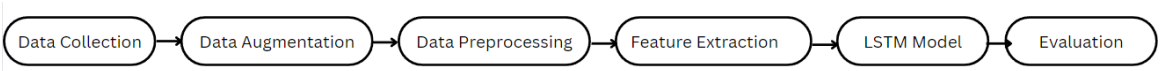

Figure 5: Architecture of the proposed model

3.2 Data Collection

This report outlines the data collection process for the fake news detection project using the WELFake dataset. The WELFake dataset was obtained from a publicly available source specifically designed for the study of fake news detection. The articles in the dataset originate from various news outlets and cover a wide range of topics, ensuring a diverse representation of real and fake news. As a result, WELFake dataset comprises of 72,134 news articles, classified 35,028 as real news and 37,106 as fake news. It contains 3 columns (title, text, label) with the label 1 represent fake and 0 represent real.

Attributes	Description	Size
Feature	Title and Text	72,134 x 3 columns
Label	Real as 0 and Fake as 1	

Table 1: WELFake Dataset Description

3.3 Data Preprocessing

In this step, we use Python library NLTK to convert text to tokens, remove punctuations and stop words and lemmatization. We also use TF-IDF for feature extraction to find word frequency.

3.3.1 Data Cleaning

Data Cleaning is a foundational process in data preprocessing, essential for ensuring that datasets are accurate, consistent and ready for machine learning tasks. The process typically begins with identifying and handling missing data, where techniques

such as deletion of missing values are applied based on the model requirements. Duplicates in the dataset are removed to prevent skewing of statistical analyses or model training outcomes. In our proposed model, we combine Title and Description together into just Text, then we check if there's any null in the dataset base on Text. We then proceed to drop any other column beside Text and Label which is what we need to train our model. To ensure that in this Text column, all rows are of string type, we convert it into string before shuffle it together and use drop_duplicates on Text to make sure that we have only unique text in the dataset. We then initialize 2 NLTK tools: stopwords and WordNetLemmatizer.

3.3.2 Stop-Word Removal

Stop words are irrelevant words that are commonly used such as (this, a, the, in, and, etc.…). These stop words take up valuable processing time without contributing into training the model. In NLTK, there are 179 stop words in English language. In this step, we convert the text in each of the row in our dataset into token using word_tokenize from NLTK then proceed to turn all tokens into lowercase and then remove punctuation and stopwords from our tokens.

3.3.3 Lemmatization

Lemmatization is a method to turn any form of words into their base form or root form. For example: "walking", "walks", "walked" turn into "walk". Inflectional endings such as "s", "ed" and "ing" are removed. Lemmatization groups these words as its root form "walk". After using stopwords, we used the lemmatize function to return tokens to its base form and return a string of each text by join them all together with a space in between each word.

3.4 TF-IDF

Feature extraction is a crucial step in machine learning pipeline, transforming raw data into a format that is suitable for modeling. We used TF-IDF (Term Frequency – Inverse Document Frequency), which is a technique in NLP for quantifying the importance of words in a document relative to a corpus. This method allowed the model to calculates the term frequency (TF) of each word, which measures how frequently a word appears in a document, then computes the inverse document frequency (IDF), which reduces the weight of rare words that are more informative. Finally, by multiplying TF

and IDF, the model obtains the TF-IDF score for each word, capturing both its importance and uniqueness across the corpus.

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

Table 2: Example of the calculation of TF-IDF

After calculating the TF-IDF scores, the text data is transformed into a sparse matrix, where each row represents a document and each column represents a word, with the matrix entries indicating the TF-IDF score of each word in each document. This matrix serves as the feature set for machine learning models, enabling them to learn from the weighted importance of words.

3.5 LSTM

We built our model on deep learning LSTM model, which is well-suited for sequential data processing due to its ability to capture long-term dependencies and contextual information. The architecture of our LSTM model is shown below and comprises 6 layers:

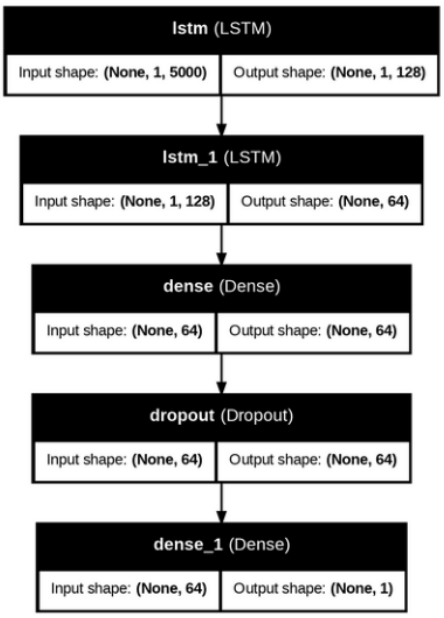

Figure 6: LSTM Model

- **Input Layer:** Accepts input sequences with a specified shape, serving as the entry point for the model.
- **First LSTM Layer:** Processes the input sequences, maintaining a memory of previous steps, and capturing contextual information, with 128 units and the return of full sequences for subsequent layers.
- **Second LSTM Layer:** Further processes the sequences, consolidating the information with 64 units.

- **Dense Layer:** Adds 64 units with a ReLU activation function to introduce non-linearity and capture complex patterns in the data.
- **Dropout Layer:** Randomly drops 50% of the input units during training to prevent overfitting and encourage the network to learn robust features that generalize well to unseen data.
- **Output Dense Layer:** Produces a single output, representing the probability that the input text is fake news. The sigmoid activation function ensures the output is between 0 and 1, making it suitable for binary classification.

This model is then compiled with Adam optimizer and Binary Cross-entropy loss function. The Adam optimizer is chosen for its efficiency and ability to handle sparse gradients, while Binary Cross-entropy loss function is appropriate for binary classification tasks. Specifically, the model uses the 'BinaryCrossentropy' loss with 'from_logits=True' to ensure numerical stability and proper gradient calculations when dealing with logits. The learning rate is set to 1e-4 to facilitate gradual convergence during training. Additionally, the model's performance is evaluated using accuracy as the metric.

3.6 Evaluation

The datasets were split into 3 parts, training 70%, testing 15% and validation 15%. Once training has been completed, the testing dataset can be used to evaluate the model. We used a confusion matrix to evaluate the performance of fake news classification.

Confusion Matrix is a graph that is used to define the performance of a model on test data, which consists of: true positive, true negative, false positive and false negative. The concepts of these are classified as follows:

- True Positive (TP): when the model predicts TRUE where the news is actually TRUE.
- True Negative (TN): when the model predicts FAKE where the news is actually FAKE.
- False Positive (FP): when the model predicts TRUE where the news is actually FAKE.
- False Negative (FN): when the model predicts FAKE where the news is actually TRUE.

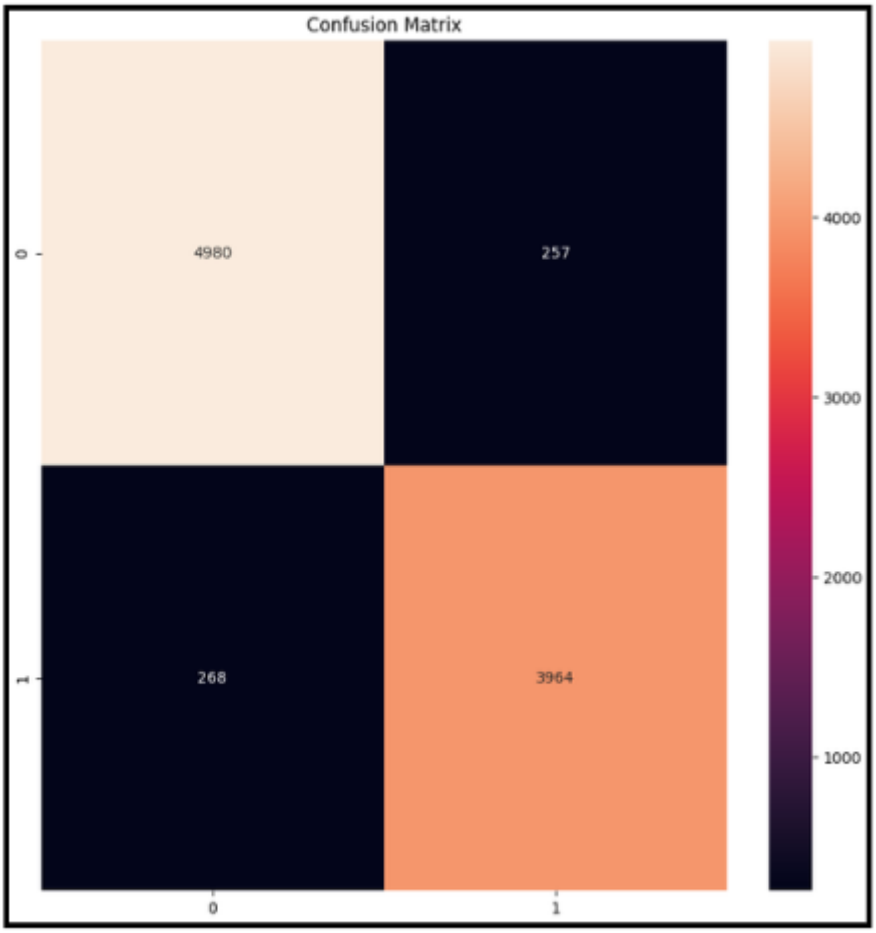

Figure 7: Confusion Matrix

Following the confusion matrix result can be evaluated using the classification measure, an extended version of confusion matrix including other measures that can help achieve better understanding of our model performance.

3.6.1 Accuracy

Accuracy measures the proportion of correct predictions (both true positives and true negatives) out of the total number of predictions. Accuracy is a useful metric when the classes are balanced. However, it can be misleading in cases of imbalanced datasets.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

3.6.2 Precision

Precision (also called Positive Predictive Value) measures the proportion of correct positive predictions out of the total predicted positives. Precision is important when the cost of false positives is high. It tells us how many of the predicted positive instances are actually positive.

$$\text{Precision} = \frac{TP}{TP+FP}$$

3.6.3 Recall

Recall (also called Sensitivity or True Positive Rate) measures the proportion of correct positive predictions out of the actual positives. Recall is important when the cost of false negatives is high. It tells us how many of the actual positive instances are correctly predicted.

$$\text{Recall} = \frac{TP}{TP+FN}$$

3.6.4 F1 Score

F1 Score is the harmonic mean of Precision and Recall, providing a single metric that balances both concerns. F1 Score is useful when you need a balance between Precision and Recall, especially in cases of imbalanced datasets. It gives a better measure of the incorrectly classified cases than the Accuracy Metric.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

CHAPTER 4
IMPLEMENTATION

4.1 Data Preprocessing

During this first step, we start by loading our dataset from MongoDB using pandas to turn the data into DataFrame. We didn't need to drop null as MongoDB ignores empty string.

```
from pymongo import MongoClient

uri = "mongodb+srv://thamveasna123:BlazinfuryX0@fakenewsdetection.hyay8j1.mongodb.net/"
client = MongoClient(uri)
db = client.Fake_News_Detection
mixed_news_collection = db['Mixed_News']
mixed = mixed_news_collection.find()
df= pd.DataFrame(mixed)
```

We then combine 'title' and 'description' into a single column called 'text' for easier processing and to make sure that all rows are string to avoid implication is the process.

```
df['text'] = df['title'] + df['description']
df['text'] = df['text'].astype(str)
```

After getting column 'text', We drop 'title', 'description' and '_id' generated by MongoDB, then we drop duplicates just in case of having duplicate news.

```
df.drop(['title', 'description', '_id'], axis=1, inplace=True)
df = df.drop_duplicates(subset=['text'])
```

To make sure that both fake and real news are random, we decided to shuffle the data frame.

```
df = shuffle(df).reset_index(drop=True)
```

We initialize NLTK tools such as stop words and lemmatizer.

```
# Initialize NLTK tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

We define a function called 'preprocess'. This function will tokenize our text into token with word_tokenize from NLTK and convert all those tokens into lowercase, then it's going to remove punctuation and stop word and lemmatize the token with NLTK tools we initialize earlier. Then apply it onto our text.

```
def preprocess(text):
    # Tokenize the text
    tokens = word_tokenize(text)

    # Convert to lower case
    tokens = [token.lower() for token in tokens]

    # Remove punctuation and stop words
    tokens = [token for token in tokens if token.isalnum() and token not in stop_words]

    # Lemmatize tokens
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Join tokens back to string
    return ' '.join(tokens)

df['text'] = df['text'].apply(preprocess)
```

4.2 Feature Extraction

Before we proceed to do feature extraction we split our data into 70% train data, 15% test data, and 15% validation data.

```
# Step 1: Split the data into 70% training and 30% remaining (validation + test)
X_train, temp_data, y_train, temp_labels = train_test_split(df['text'], df['label'], test_size=0.3, random_state=42)

# Step 2: Split the remaining 30% into 15% validation and 15% test
X_val, X_test, y_val, y_test = train_test_split(temp_data, temp_labels, test_size=0.5, random_state=42)
```

We want to turn our text data into numerical feature vectors with max features is up to the top 5000 based on their TF-IDF score across the entire dataset. By focusing on the top 5000 terms, we reduce the dimensionality of the feature space which improved training speed and avoid overfitting.

Then we fit this vectorizer into our train, test and validation text to be ready for model training and testing.

```
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
X_val_tfidf = tfidf_vectorizer.transform(X_val)
```

Then we convert them to array and reshape the array into 3 dimensions.

```
X_train_tfidf = X_train_tfidf.toarray().reshape((X_train_tfidf.shape[0], 1, X_train_tfidf.shape[1]))
X_test_tfidf = X_test_tfidf.toarray().reshape((X_test_tfidf.shape[0], 1, X_test_tfidf.shape[1]))
X_val_tfidf = X_val_tfidf.toarray().reshape((X_val_tfidf.shape[0], 1, X_val_tfidf.shape[1]))
```

The output of our TF-IDF vectorizer is in sparse matrices. While sparse matrices are memory-efficient for storage, our models expect the input data to be in dense format. In

reshape ‘X_train_tfidf.shape[0]’ means the number of samples in the dataset. 1 means we add a new dimension of size 1 and ‘X_train_tfidf.shape[1]’ means the number of features. The same goes for test and validation data.

4.3 Model Training

We train our model using Keras, a sequential model to create a linear stack of layers. It allows us to build models layer by layer, where each layer has exactly one input tensor and one output tensor. In our sequential model, we have 6 layers:

```
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(1, 5000)),
    tf.keras.layers.LSTM(128, return_sequences=True),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

1. Input Layer: Define the shape of the input data. It tells the model that each input will be a 2D array with shape (1, 5000) where 1 is the sequence length or a single time step and 5000 is the number of features

2. First LSTM Layer: We define the number of units for LSTM in this layer to be 128 and ensure that the output is a sequence (3D tensor) for stacking another LSTM layer.

3. Second LSTM Layer: We define another LSTM with 64 units without return_sequences for reducing the dimensionality of the data passed to the next layer

4. Dense Layer: This layer is where each input neuron is connected to each output neuron. We define the number of neurons to be 64 and activate function ‘relu’ (Rectified Linear Unit), which introduce non-linearity into the model.

5. Dropout Layer: This layer prevent overfitting by randomly setting a fraction (50% in this case) of input units to 0 at each update during training.

6. Output Dense Layer: This layer is the output layer for binary classification problem. With activation function ‘sigmoid’ the result will return a range between 0 and 1, which is suitable for our study.

Below is our 6 layers sequential model summary with LSTM. We get a total of 2,679,681 trainable parameters.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 1, 128)	2,626,048
lstm_3 (LSTM)	(None, 64)	49,408
dense_2 (Dense)	(None, 64)	4,160
dropout_1 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

Total params: 2,679,681 (10.22 MB)
Trainable params: 2,679,681 (10.22 MB)
Non-trainable params: 0 (0.00 B)

Figure 8: Model Summary

We compile the model with BinaryCrossentropy loss function because it measures the performance of classification model whose output is a probability value between 0 and 1 like our fake news classification and Adam (Adaptive Moment Estimation) optimizer that combines the advantages of 2 other extensions of stochastic gradient descent, namely AdaGrad and RMSProp with learning rate of '1e-4' (0.0001) represent the step size of each iteration to get as much accuracy as possible.

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

In order to not overtrain and get the best possible model from our training I use EarlyStopping to monitor on validation loss with patience = 3 just in case validation loss would improve in later epoch.

```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

Then we fit it into our model with 10 iterations, validate with our validation data that we split earlier, batch size of 64. As we have more computational resources, we have tried with batch size of 32, 64, 128, 256, 512 and found out that the best size for our model is 64. We have the model shuffle the training data per epoch to prevent overfitting and callbacks for using early stop for best model.

```
history = model.fit(
    X_train_tfidf,
    y_train,
    epochs=10,
    validation_data=(X_val_tfidf, y_val),
    batch_size=128,
    shuffle=True,
    callbacks=[early_stop]
    )
```

CHAPTER 5
RESULTS AND DISCUSSION

5.1 Training Result

After we have done our model training, we have found that our model train up to epoch 9/10 with the best model at epoch 7/10 with training loss of 11.05%, training accuracy of 96.24%, validation loss of 16.27 % and validation accuracy of 94.12%. In this case, the model is performing really well with high accuracy and low loss on both training and validation datasets, indicating effective learning without significant overfitting.

```
Epoch 1/10
346/346 ━━━━━━━━━━━━━━━━━━━━ 29s 74ms/step - accuracy: 0.5997 - loss: 0.6745 - val_accuracy: 0.8497 - val_loss: 0.4455
Epoch 2/10
346/346 ━━━━━━━━━━━━━━━━━━━━ 23s 67ms/step - accuracy: 0.8799 - loss: 0.3583 - val_accuracy: 0.9168 - val_loss: 0.2288
Epoch 3/10
346/346 ━━━━━━━━━━━━━━━━━━━━ 41s 67ms/step - accuracy: 0.9275 - loss: 0.2031 - val_accuracy: 0.9306 - val_loss: 0.1870
Epoch 4/10
346/346 ━━━━━━━━━━━━━━━━━━━━ 40s 65ms/step - accuracy: 0.9448 - loss: 0.1578 - val_accuracy: 0.9356 - val_loss: 0.1715
Epoch 5/10
346/346 ━━━━━━━━━━━━━━━━━━━━ 22s 62ms/step - accuracy: 0.9522 - loss: 0.1364 - val_accuracy: 0.9390 - val_loss: 0.1697
Epoch 6/10
346/346 ━━━━━━━━━━━━━━━━━━━━ 43s 67ms/step - accuracy: 0.9572 - loss: 0.1194 - val_accuracy: 0.9390 - val_loss: 0.1634
Epoch 7/10
346/346 ━━━━━━━━━━━━━━━━━━━━ 41s 67ms/step - accuracy: 0.9624 - loss: 0.1105 - val_accuracy: 0.9412 - val_loss: 0.1627
Epoch 8/10
346/346 ━━━━━━━━━━━━━━━━━━━━ 40s 66ms/step - accuracy: 0.9679 - loss: 0.0964 - val_accuracy: 0.9425 - val_loss: 0.1688
Epoch 9/10
346/346 ━━━━━━━━━━━━━━━━━━━━ 41s 65ms/step - accuracy: 0.9708 - loss: 0.0923 - val_accuracy: 0.9416 - val_loss: 0.1669
```

Figure 9: Model Result

5.1.1 Model Accuracy

Accuracy is the ratio of correctly predicted instances to the total instances in the dataset. Our model has achieved a high accuracy of over 90% on both training data and validation data.

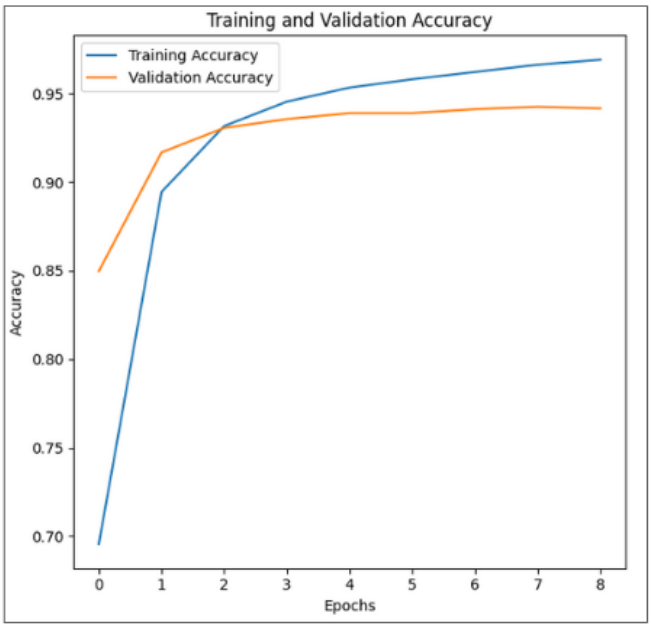

Figure 10: Training and Validation Accuracy

5.1.2 Model Loss

Loss is the difference between the predicted values and the actual values, serving as a measure of the model's prediction error. Our model has achieved a loss of 11.05% for training loss and 16.27% for validation loss.

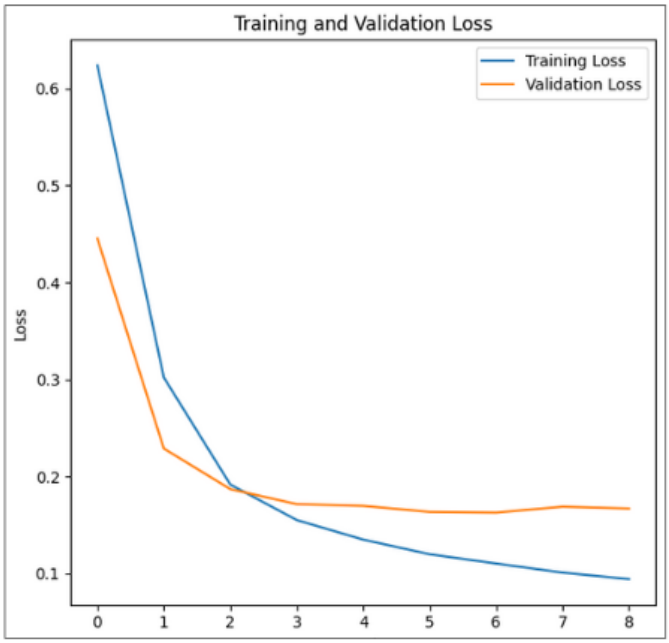

Figure 11: Training and Validation Loss

5.2 Model Evaluation

We have used testing data split from our dataset to evaluate for seen dataset and another dataset from Kaggle for unseen dataset. The new dataset has 44,898 news articles with 23,481 being fake news and 21,417 being real news.

Metric	Testing Data	Unseen Data
Accuracy	93.58%	97.46%
Precision	93.28%	99.37%
Recall	92.52%	95.74%
F1-Score	92.89%	97.52%

Table 3: Comparison of Testing Data and Unseen Data

The presented results show the comparison of model prediction on both testing dataset and unseen dataset based on evaluation metrics. From the table above, we can see that model gives out similar result for both testing and unseen data which means that it works exceptionally well.

5.3 Discussion

The overall testing accuracy align perfectly well with the training and validation accuracy as well as precision, recall and F1-Score. This means that our model can deal and generalizes unseen data very incredibly well without having any significant drop in performance in any of the test so far. Compare to IFND, we have achieved a 1.57% higher than their highest accuracy machine learning classifier, 2.86% higher than their LSTM model and 2.87% higher than their Bi-LSTM model.

CHAPTER 6
CONCLUSION

6.1 Conclusion

The rise of fake news in the digital age presents a major challenge for people and society. Misinformation spreads quickly on social media and websites, eroding trust, distorting facts, and creating widespread misunderstandings. In this study, we proposed a LSTM model to counterattack this problem with the help of powerful toolkit NLTK and TF-IDF as feature extraction for fake news classification. Experiment result shows that the overall testing accuracy of 94.45% align well with the training accuracy of 95.57% and validation accuracy of 93.78%. The testing data shows a high precision of 93.95% and recall of 93.62%. This means that the model is good at identifying both fake and real news and generalizes the unseen data in the test data reasonably well. To ensure that our model is working well, we applied new unseen datasets of news articles without labels published on Kaggle to test our model. The new unseen dataset had a high accuracy of 97.46%, precision of 99.37%, recall of 95.74% and f1-score of 97.52% compare to testing dataset. In conclusion, the LSTM model shows incredibly good results on training, validation and testing with minimal losses.

6.2 Future Work

Looking forward, we aim to train our model on Khmer datasets for fake news classification, by getting news from official news website as real news and news spreading around in social media as fake news. We would like to experiment with a different approach of combining algorithms to make up for each algorithm disadvantages and add image to the classification to improve it further.

REFERENCES

Sharma, D. K., & Garg, S. (2021). IFND: a benchmark dataset for fake news detection. Complex & Intelligent Systems, 9(3), 2843–2863. https://doi.org/10.1007/s40747-021-00552-1

Shu, K., Sliva, A., Wang, S., Tang, J., & Liu, H. (2017). Fake News Detection on Social Media. SIGKDD Explorations, 19(1), 22–36. https://doi.org/10.1145/3137597.3137600

Sharma, V. (2020). Machine Learning Algorithms via Detection of Fake News. International Journal for Research in Applied Science and Engineering Technology, 8(6), 780–784. https://doi.org/10.22214/ijraset.2020.6125

Fake News Classification. (2023, October 8). Kaggle. https://www.kaggle.com/datasets/saurabhshahane/fake-news-classification

Fake News Detection. (2023, December 17). Kaggle. https://www.kaggle.com/datasets/bhavikjikadara/fake-news-detection

Verma, P. K., Agrawal, P., Amorim, I., & Prodan, R. (2021). WELFake: Word Embedding Over Linguistic Features for Fake News Detection. IEEE Transactions on Computational Social Systems, 8(4), 881–893. https://doi.org/10.1109/tcss.2021.3068519

Shah, M. N., & Ganatra, A. (2022). A systematic literature review and existing challenges toward fake news detection models. Social Network Analysis and Mining, 12(1). https://doi.org/10.1007/s13278-022-00995-5