

Assignment 4

Udit Kumar B

TEAM ID - PNT2022TMID54091

1. Download the Dataset

<https://www.kaggle.com/code/kredy10/simple-lstm-for-text-classification/data>

2. Import the required libraries

```
import os import re
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords from
nltk.stem import WordNetLemmatizer from
wordcloud import WordCloud import
matplotlib.pyplot as plt import
tensorflow as tf
from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Dense, LSTM, Dropout, Embedding from
tensorflow.keras.callbacks import EarlyStopping from
tensorflow.keras.preprocessing.text import Tokenizer import keras
from sklearn.preprocessing import LabelEncoder from
sklearn.feature_extraction.text import TfidfVectorizer from
sklearn.model_selection import train_test_split from
google.colab import drive
```

```
#Mount and access drive
drive.mount('/content/drive', force_remount=True)
os.chdir('/content/drive/My Drive')
print("Change successful.")
```

Mounted at /content/drive Change
successful.

3. Read the dataset and do pre-processing

```
spam_df = pd.read_csv(filepath_or_buffer='Dataset-3_Spam.csv', delimiter=',', encoding='l
spam_df.head()
```

0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
#List the column names spam_df.columns
```

```
Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')
```

```
#Drop the unnamed columns
spam_df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)
spam_df.columns
```

```
Index(['v1', 'v2'], dtype='object')
```

```
#Print the number of rows in the dataset spam_df.shape
```

```
(5572, 2)
```

```
#Get the summary statistics of the dataset spam_df.describe()
```

	v1	v2
count	5572	5572
unique	5169	2
top	ham	Sorry, I'll call later
freq	4825	30

```
#Check for null values spam_df.isna().sum()
```

```
v1    0
v2    0
dtype: int64
```

```
#Check for duplicated rows spam_df.duplicated().sum()
```

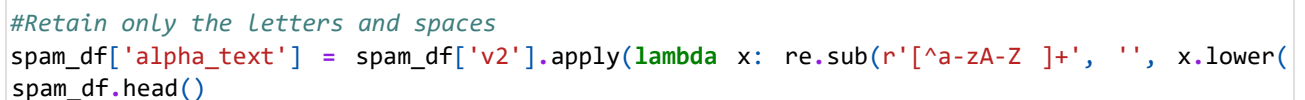
```
403
```

```
#Remove the duplicated rows spam_df = spam_df.drop_duplicates()
spam_df.duplicated().sum()
```

```
#Display the count of spam and ham labels
#Stratified-split is required
spam_df['v1'].hist(bins=3)
```

Age Group	Number of People
18-24	4500

```
#Plot the word-cloud before removing stopwords, performing Lemmatization wordcloud_vis('v2')
```



	v1	v2	alpha_text
0	ham	Go until jurong point, crazy.. Available only ...	go until jurong point crazy available only in ...
1	ham	Ok lar... Joking wif u oni... ok lar joking wif u oni	
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	free entry in a wkly comp to win fa cup final...
3	ham	U dun say so early hor... U c already then say...	u dun say so early hor u c already then say
4	ham	Nah I don't think he goes to usf, he lives aro...	nah i dont think he goes to usf he lives aroun...

```
#Remove stop-words nltk.download('stopwords')
spam_df['imp_text'] = spam_df['alpha_text'].apply(lambda x : ' '.join([word for word in x.split() if word not in stopwords]))
spam_df.head()
```

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.

	v1	v2	alpha_text	imp_text
0	ham	Go until jurong point, crazy.. Available only ...	go until jurong point crazy available only in ...	go jurong point crazy available bugis n great ...
1	ham	Ok lar... Joking wif u oni...	ok lar joking wif u oni	ok lar joking wif u oni
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	free entry in a wkly comp to win fa cup final...	free entry wkly comp win fa cup final tkts st ...
3	ham	U dun say so early hor... U c already then say...	u dun say so early hor u c already then say	u dun say early hor u c already say
4	ham	Nah I don't think he goes to usf, he lives aro...	nah i dont think he goes to usf he lives aroun...	nah dont think goes usf lives around though

```
#Tokenize the data def
tokenize(data):
    generated_token = list(data.split())
    return generated_token
spam_df['token_text'] = spam_df['imp_text'].apply(lambda x: tokenize(x))
spam_df.head()
```

	v1	v2	alpha_text	imp_text	token_text
0	ham	Go until jurong point, available bugis n great crazy.. Available only ...	go until jurong point crazy available only in ...	go jurong point crazy crazy available only in ...	[go, jurong, point, crazy, available, bugis, n... ...]
1	ham	Ok lar... Joking wif u ok lar joking wif u oni oni...	ok lar joking wif u oni oni...	ok lar joking wif u oni oni...	[ok, lar, joking, wif, u, ham oni]

Free entry in 2 a wkly free entry in a wkly free entry wkly comp [free, entry, wkly, comp,

2 spam comp to win FA Cup comp to win fa cup win fa cup final tkts st ... win, fa, cup, final,...

final...

final...

3 U dun say so early hor... u dun say so early hor u u dun say early hor u c [u, dun, say, early, hor, ham
U c already then say... c already then say already say u, c, already, say]

Nah I don't think he nah i dont think he goes nah dont think goes usf [nah, dont, think, goes, 4 ham
goes to usf, he lives to usf he lives aroun... lives around though usf, lives, around, t...
aro...

```
#Perform Lemmatization
nltk.download('wordnet')
nltk.download('omw-1.4')
lemmatizer = WordNetLemmatizer()
def lemmatization(list_of_words):
    lemmatized_list = [lemmatizer.lemmatize(word) for word in list_of_words] return
    lemmatized_list
spam_df['lemmatized_text'] = spam_df['token_text'].apply(lambda x: lemmatization(x))
spam_df.head()
```

[nltk_data] Downloading package wordnet to /root/nltk_data... [nltk_data]
Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...

	v1	v2	alpha_text	imp_text	token_text	lemmatized_text
0	ham	Go until jurong point, crazy.. Available only in ...	go until jurong point crazy available only in ...	go jurong point crazy available bugis n great ...	[go, jurong, point, crazy, available, crazy, available, bugis, n...	[go, jurong, point, crazy, available, bugis, n...
1	ham	Ok lar... Joking wif u oni...	ok lar joking wif u oni	ok lar joking wif u oni	[ok, lar, joking, wif, u, oni]	[ok, lar, joking, wif, u, oni]
2	spam	Free entry in 2 a wkly comp to win FA Cup final... final tkts st ... cup, final,...	free entry in a wkly comp to win comp win fa cup comp, win, fa, win, fa, cup, final,...	free entry wkly comp, win, fa, win, fa, cup, final,...	[free, entry, wkly, [free, entry, wkly, comp, win FA Cup fina... fa cup	[free, entry, wkly, [free, entry, wkly, comp, win FA Cup fina... fa cup
3	ham	U dun say so early hor... U c already then	u dun say so early hor u c already then say	u dun say early hor u c already then say say	[u, dun, say, early, hor, u, c, already, then say say]	[u, dun, say, early, hor, u, c, already, then say say]

Nah I don't think nah i dont think nah dont think [nah, dont, think, [nah, dont, think, go, 4 ham he goes to usf, he goes to usf he goes usf lives goes, usf, lives, usf, life, around, he lives aro... lives aroun... around though around, t... though]

```
#Combine the tokens (into sentences) to get the final cleansed data spam_df['clean'] =
spam_df['lemmatized_text'].apply(lambda x: ' '.join(x)) spam_df.head()
```

	v1	v2	alpha_text	imp_text	token_text	lemmatized_text	clean
--	----	----	------------	----------	------------	-----------------	-------



```

#Number of unique words in spam and ham df1
= spam_df.loc[spam_df['v1'] == 'spam'] df2
= spam_df.loc[spam_df['v1'] == 'ham']
spam = set()
df1['clean'].str.lower().str.split().apply(spam.update)
print("Number of unique words in spam", len(spam))
ham = set()
df2['clean'].str.lower().str.split().apply(ham.update)
print("Number of unique words in ham", len(ham))

```

Number of unique words in spam 2037

Number of unique words in ham 6738

```

#Find the number of overlapping words between spam and ham labels print("Number of
overlapping words between spam and ham: ", len(spam & ham))

```

Number of overlapping words between spam and ham: 895

```

#Maximum number of words in a sentence
#Useful for applying padding
spam_df['clean'].apply(lambda x:len(str(x).split())).max()

```

80

```

#Prepare the data for training X = spam_df['clean'] y =
spam_df['v1']

```

```

#Convert the class labels into integer values le = LabelEncoder() y =
le.fit_transform(y) y

```

array([0, 0, 1, ..., 0, 0, 0])

X.shape

(5169,)

y.shape

(5169,)

```

#Split the data into train, test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=4

```



```
tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(X_train)
tokenized_train = tokenizer.texts_to_sequences(X_train)
X_train = tf.keras.utils.pad_sequences(tokenized_train, maxlen=100)
```

```
tokenized_test = tokenizer.texts_to_sequences(X_test)
X_test = tf.keras.utils.pad_sequences(tokenized_test, maxlen=100)
```

4. Create the model

```
#Create a wrapper to add layers to the model model = Sequential()
```

5. Add Layers (LSTM, Dense-(Hidden Layers), Output)

```
model.add(Embedding(1000, output_dim=50, input_length=100))
model.add(LSTM(units=64, return_sequences = True, dropout = 0.2))
model.add(LSTM(units=32, dropout = 0.1)) model.add(Dense(units = 64, activation
= 'relu')) model.add(Dense(units = 32, activation = 'relu')) model.add(Dense(1,
activation='sigmoid'))
```

```
model.summary()
```

Model: "sequential_12"

Layer (type)	Output Shape	Param #
embedding_14 (Embedding)	(None, 100, 50)	50000
lstm_38 (LSTM)	(None, 100, 64)	29440
lstm_39 (LSTM)	(None, 32)	12416
dense_25 (Dense)	(None, 64)	2112
dense_26 (Dense)	(None, 32)	2080
dense_27 (Dense)	(None, 1)	33

Total params: 96,081
Trainable params: 96,081
Non-trainable params: 0

6. Compile the Model


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

7. Fit the Model

```
model.fit(X_train, y_train, batch_size=128, epochs=10, validation_split=0.2, callbacks=[Earl
```

```
Epoch 1/10
28/28 [=====] - 13s 308ms/step - loss: 0.4777 - accuracy: 0.8603 -
val_loss: 0.3748 - val_accuracy: 0.8760
Epoch 2/10
28/28 [=====] - 8s 272ms/step - loss: 0.3768 - accuracy: 0.8731
- val_loss: 0.3598 - val_accuracy: 0.8760
Epoch 3/10
28/28 [=====] - 8s 271ms/step - loss: 0.2600 - accuracy: 0.8990 -
val_loss: 0.1339 - val_accuracy: 0.9647
Epoch 4/10
28/28 [=====] - 8s 272ms/step - loss: 0.0874 - accuracy: 0.9772
- val_loss: 0.0870 - val_accuracy: 0.9738
Epoch 5/10
28/28 [=====] - 8s 271ms/step - loss: 0.0602 - accuracy: 0.9829
- val_loss: 0.0748 - val_accuracy: 0.9761
Epoch 6/10
28/28 [=====] - 7s 268ms/step - loss: 0.0520 - accuracy: 0.9843
- val_loss: 0.0687 - val_accuracy: 0.9772
Epoch 7/10
28/28 [=====] - 8s 270ms/step - loss: 0.0350 - accuracy: 0.9898 -
val_loss: 0.0646 - val_accuracy: 0.9795
Epoch 8/10
28/28 [=====] - 8s 269ms/step - loss: 0.0269 - accuracy: 0.9920
- val_loss: 0.0685 - val_accuracy: 0.9761
Epoch 9/10
28/28 [=====] - 8s 272ms/step - loss: 0.0215 - accuracy: 0.9943
- val_loss: 0.0711 - val_accuracy: 0.9772
<keras.callbacks.History at 0x7f9280f9aa90>
```

8. Save the Model

```
model.save('spam-classifier.h5')
```

9. Test the Model

```
print("Accuracy of the model on Testing Data is - " , model.evaluate(X_test,y_test)[1]*1
```

25/25 [=====] - 1s 26ms/step - loss: 0.0625 - accuracy: 0.9871