**MAHENDRA ENGINEERING COLLEGE FOR WOMEN**

NAME :nandhini

CLASS :IV-IT

SUB :IBM(AI)

REG NO :611419205016

**Import the Dataset**

from google.colab import files  
uploaded = files.upload()

Saving spam.csv to spam.csv

**Import required libraries**

import csv  
import tensorflow as tf  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.preprocessing.sequence import pad\_sequences  
import nltk  
nltk.download('stopwords')   
from nltk.corpus import stopwords  
STOPWORDS = set(stopwords.words('english'))

[nltk\_data] Downloading package stopwords to /root/nltk\_data...  
[nltk\_data] Unzipping corpora/stopwords.zip.

[nltk\_data] Unzipping corpora/stopwords.zip.

**Import dataset**

import io  
dataset = pd.read\_csv(io.BytesIO(uploaded['spam.csv']), encoding = "ISO-8859-1")

dataset

v1 v2 Unnamed: 2 \  
0 ham Go until jurong point, crazy.. Available only ... NaN   
1 ham Ok lar... Joking wif u oni... NaN   
2 spam Free entry in 2 a wkly comp to win FA Cup fina... NaN   
3 ham U dun say so early hor... U c already then say... NaN   
4 ham Nah I don't think he goes to usf, he lives aro... NaN   
... ... ... ...   
5567 spam This is the 2nd time we have tried 2 contact u... NaN   
5568 ham Will Ì\_ b going to esplanade fr home? NaN   
5569 ham Pity, \* was in mood for that. So...any other s... NaN   
5570 ham The guy did some bitching but I acted like i'd... NaN   
5571 ham Rofl. Its true to its name NaN   
  
 Unnamed: 3 Unnamed: 4   
0 NaN NaN   
1 NaN NaN   
2 NaN NaN   
3 NaN NaN   
4 NaN NaN   
... ... ...   
5567 NaN NaN   
5568 NaN NaN   
5569 NaN NaN   
5570 NaN NaN   
5571 NaN NaN   
  
[5572 rows x 5 columns]

vocab\_size = 5000  
embedding\_dim = 64  
max\_length = 200  
trunc\_type = 'post'  
padding\_type = 'post'  
oov\_tok = ''  
training\_portion = .8

**Read the dataset and do pre-processing.**

**To remove the stop words.**

articles = []  
labels = []  
  
**with** open("spam.csv", 'r', encoding = "ISO-8859-1") as dataset:  
 reader = csv.reader(dataset, delimiter=',')  
 next(reader)  
 **for** row **in** reader:  
 labels.append(row[0])  
 article = row[1]  
 **for** word **in** STOPWORDS:  
 token = ' ' + word + ' '  
 article = article.replace(token, ' ')  
 article = article.replace(' ', ' ')  
 articles.append(article)  
print(len(labels))  
print(len(articles))

5572  
5572

**Train the model**

train\_size = int(len(articles) \* training\_portion)  
train\_articles = articles[0: train\_size]  
train\_labels = labels[0: train\_size]  
validation\_articles = articles[train\_size:]  
validation\_labels = labels[train\_size:]  
print(train\_size)  
print(len(train\_articles))  
print(len(train\_labels))  
print(len(validation\_articles))  
print(len(validation\_labels))

4457  
4457  
4457  
1115  
1115

tokenizer = Tokenizer(num\_words = vocab\_size, oov\_token=oov\_tok)  
tokenizer.fit\_on\_texts(train\_articles)  
word\_index = tokenizer.word\_index  
dict(list(word\_index.items())[0:10])

{'': 1,  
 'i': 2,  
 'u': 3,  
 'call': 4,  
 'you': 5,  
 '2': 6,  
 'get': 7,  
 "i'm": 8,  
 'ur': 9,  
 'now': 10}

**Training data to Sequences**

train\_sequences = tokenizer.texts\_to\_sequences(train\_articles)  
print(train\_sequences[10])

[8, 190, 37, 201, 30, 260, 293, 991, 222, 53, 153, 3815, 423, 46]

Train neural network for NLP

train\_padded = pad\_sequences(train\_sequences, maxlen=max\_length, padding=padding\_type, truncating=trunc\_type)  
print(len(train\_sequences[0]))  
print(len(train\_padded[0]))  
  
print(len(train\_sequences[1]))  
print(len(train\_padded[1]))  
  
print(len(train\_sequences[10]))  
print(len(train\_padded[10]))

16  
200  
6  
200  
14  
200

print(train\_padded[10])

[ 8 190 37 201 30 260 293 991 222 53 153 3815 423 46  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0]

validation\_sequences = tokenizer.texts\_to\_sequences(validation\_articles)  
validation\_padded = pad\_sequences(validation\_sequences, maxlen=max\_length, padding=padding\_type, truncating=trunc\_type)  
  
print(len(validation\_sequences))  
print(validation\_padded.shape)

1115  
(1115, 200)

label\_tokenizer = Tokenizer()  
label\_tokenizer.fit\_on\_texts(labels)  
  
training\_label\_seq = np.array(label\_tokenizer.texts\_to\_sequences(train\_labels))  
validation\_label\_seq = np.array(label\_tokenizer.texts\_to\_sequences(validation\_labels))  
print(training\_label\_seq[0])  
print(training\_label\_seq[1])  
print(training\_label\_seq[2])  
print(training\_label\_seq.shape)  
  
print(validation\_label\_seq[0])  
print(validation\_label\_seq[1])  
print(validation\_label\_seq[2])  
print(validation\_label\_seq.shape)

[1]  
[1]  
[2]  
(4457, 1)  
[1]  
[2]  
[1]  
(1115, 1)

reverse\_word\_index = dict([(value, key) **for** (key, value) **in** word\_index.items()])  
  
**def** decode\_article(text):  
 **return** ' '.join([reverse\_word\_index.get(i, '?') **for** i **in** text])  
print(decode\_article(train\_padded[10]))  
print('---')  
print(train\_articles[10])

i'm gonna home soon want talk stuff anymore tonight k i've cried enough today ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?  
---  
I'm gonna home soon want talk stuff anymore tonight, k? I've cried enough today.

**To implement LSTM**

model = tf.keras.Sequential([  
   
 tf.keras.layers.Embedding(vocab\_size, embedding\_dim),  
 tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(embedding\_dim)),  
 tf.keras.layers.Dense(embedding\_dim, activation='relu'),  
 tf.keras.layers.Dense(6, activation='softmax')  
])  
model.summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 embedding (Embedding) (None, None, 64) 320000   
   
 bidirectional (Bidirectional (None, 128) 66048   
 l)   
   
 dense (Dense) (None, 64) 8256   
   
 dense\_1 (Dense) (None, 6) 390   
   
=================================================================  
Total params: 394,694  
Trainable params: 394,694  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

print(set(labels))

{'spam', 'ham'}

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  
num\_epochs = 10  
history = model.fit(train\_padded, training\_label\_seq, epochs=num\_epochs, validation\_data=(validation\_padded, validation\_label\_seq), verbose=2)

Epoch 1/10  
140/140 - 37s - loss: 0.3177 - accuracy: 0.9251 - val\_loss: 0.0387 - val\_accuracy: 0.9830 - 37s/epoch - 265ms/step  
Epoch 2/10  
140/140 - 35s - loss: 0.0310 - accuracy: 0.9915 - val\_loss: 0.0318 - val\_accuracy: 0.9901 - 35s/epoch - 252ms/step  
Epoch 3/10  
140/140 - 32s - loss: 0.0130 - accuracy: 0.9975 - val\_loss: 0.0627 - val\_accuracy: 0.9857 - 32s/epoch - 230ms/step  
Epoch 4/10  
140/140 - 31s - loss: 0.0060 - accuracy: 0.9987 - val\_loss: 0.0478 - val\_accuracy: 0.9901 - 31s/epoch - 220ms/step  
Epoch 5/10  
140/140 - 30s - loss: 0.0042 - accuracy: 0.9989 - val\_loss: 0.0613 - val\_accuracy: 0.9883 - 30s/epoch - 215ms/step  
Epoch 6/10  
140/140 - 29s - loss: 0.0033 - accuracy: 0.9991 - val\_loss: 0.0728 - val\_accuracy: 0.9883 - 29s/epoch - 210ms/step  
Epoch 7/10  
140/140 - 29s - loss: 0.0020 - accuracy: 0.9996 - val\_loss: 0.0540 - val\_accuracy: 0.9865 - 29s/epoch - 208ms/step  
Epoch 8/10  
140/140 - 31s - loss: 7.6466e-04 - accuracy: 0.9998 - val\_loss: 0.0644 - val\_accuracy: 0.9901 - 31s/epoch - 219ms/step  
Epoch 9/10  
140/140 - 30s - loss: 3.9159e-04 - accuracy: 1.0000 - val\_loss: 0.0678 - val\_accuracy: 0.9883 - 30s/epoch - 211ms/step  
Epoch 10/10  
140/140 - 29s - loss: 1.7514e-04 - accuracy: 1.0000 - val\_loss: 0.0726 - val\_accuracy: 0.9883 - 29s/epoch - 208ms/step

**def** plot\_graphs(history, string):  
 plt.plot(history.history[string])  
 plt.plot(history.history['val\_'+string])  
 plt.xlabel("Epochs")  
 plt.ylabel(string)  
 plt.legend([string, 'val\_'+string])  
 plt.show()  
   
plot\_graphs(history, "accuracy")  
plot\_graphs(history, "loss")

