

## Problem Statement:- SMS SPAM Classification

Over recent years, as the popularity of mobile phone devices has increased, Short Message Service (SMS) has grown into a multi-billion dollar industry. At the same time, reduction in the cost of messaging services has resulted in growth in unsolicited commercial advertisements (spams) being sent to mobile phones. Due to Spam SMS, Mobile service providers suffer from some sort of financial problems as well as it reduces calling time for users. Unfortunately, if the user accesses such Spam SMS they may face the problem of virus or malware. When SMS arrives at mobile it will disturb mobile user privacy and concentration. It may lead to frustration for the user. So Spam SMS is one of the major issues in the wireless communication world and it grows day by day.

### ▼ Solution:

#### Step1 : Upload the dataset

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

Choose Files No file chosen

this cell to enable.

Saving spam (1).csv to spam (1).csv

Upload widget is only available when the cell has been executed in

### ▼ Step 2: Importing Required Libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
%matplotlib inline
import csv
```

```
import tensorflow as tf
import pandas as pd
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.
```

▼ **Step3:Reading the dataset and pre-processing the data**

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

dataset
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
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
...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	NaN	NaN	NaN
5568	ham	Will Ì_ b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other s...	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd...	NaN	NaN	NaN
5571	ham	Rofl. Its true to its name	NaN	NaN	NaN

5572 rows × 5 columns

▼ **Step 4:Create Model**

```
vocab_size = 5000
```

```

embedding_dim = 64
max_length = 200
trunc_type = 'post'
padding_type = 'post'
oov_tok = ''
training_portion = .8

articles = []
labels = []

with open("spam (1).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

```

## ▼ Step 5: Add Layers (LSTM, Dense-(Hidden Layers), Output)

```

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}
```

## StepFit the Model

```
train_sequences = tokenizer.texts_to_sequences(train_articles)
print(train_sequences[10])
```

```
[8, 189, 37, 201, 30, 260, 293, 991, 222, 53, 153, 3815, 423, 46]
```

```
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding=padding_type, truncate='left')
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])
```

```
[ 8 189 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)

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.
```

```
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
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'}
```

## ▼ Step 6: Compile The Model

```
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=(val

Epoch 1/10
140/140 - 21s - loss: 0.3154 - accuracy: 0.9295 - val_loss: 0.0493 - val_accuracy: 0.98
Epoch 2/10
140/140 - 18s - loss: 0.0287 - accuracy: 0.9910 - val_loss: 0.0306 - val_accuracy: 0.99
Epoch 3/10
140/140 - 18s - loss: 0.0126 - accuracy: 0.9971 - val_loss: 0.0400 - val_accuracy: 0.98
```

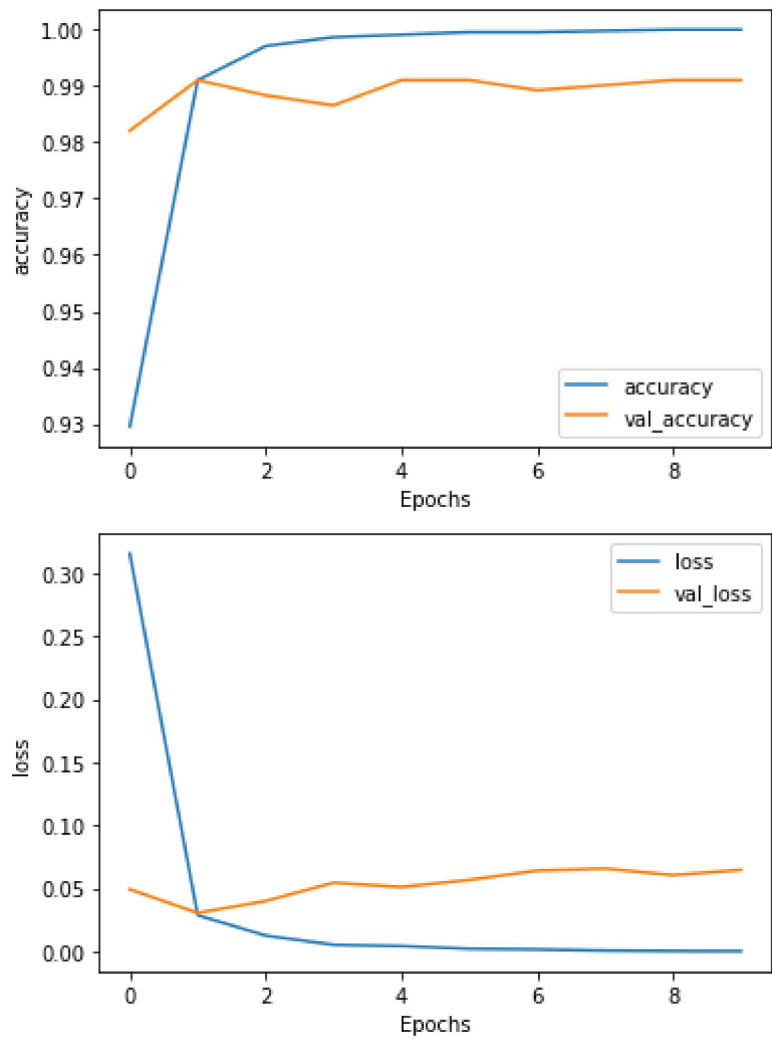
```
Epoch 4/10
140/140 - 18s - loss: 0.0053 - accuracy: 0.9987 - val_loss: 0.0545 - val_accuracy: 0.98
Epoch 5/10
140/140 - 18s - loss: 0.0043 - accuracy: 0.9991 - val_loss: 0.0511 - val_accuracy: 0.99
Epoch 6/10
140/140 - 18s - loss: 0.0021 - accuracy: 0.9996 - val_loss: 0.0568 - val_accuracy: 0.99
Epoch 7/10
140/140 - 19s - loss: 0.0016 - accuracy: 0.9996 - val_loss: 0.0641 - val_accuracy: 0.98
Epoch 8/10
140/140 - 18s - loss: 6.9357e-04 - accuracy: 0.9998 - val_loss: 0.0657 - val_accuracy:
Epoch 9/10
140/140 - 18s - loss: 2.7686e-04 - accuracy: 1.0000 - val_loss: 0.0605 - val_accuracy:
Epoch 10/10
140/140 - 18s - loss: 1.4492e-04 - accuracy: 1.0000 - val_loss: 0.0646 - val_accuracy:
```



## ▼ Step 7: Test The Model

```
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")
```



Colab paid products - [Cancel contracts here](#)