# ASSIGNMENT 4

| | |
|---|---|
| ASSIGNMENT DATE | 8 OCTOBER 2020 |
| STUDENT NAME | JAYARAJ D |
| STUDENT ROLL NUMBER | 2019504530 |
| MAXIMUM MARKS | 2 MARKS |

# Assignment 4

# Tasks

Perform the Below Tasks to complete the assignment:-

- Download the Dataset:- Dataset
- Import required library
- Read dataset and do pre-processing
- Create Model
- Add Layers (LSTM, Dense-(Hidden Layers), Output)
- Compile the Model
- Fit the Model
- Save The Model
- Test The Model

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

# 1. Load the dataset

In [ ]:
```
dataset_location = "/content/drive/MyDrive/IBM/spam.csv"
```

# 2. Import the library

In [ ]:
```
import pandas as pd
import nltk
import re
import numpy as np
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.translate.ribes_score import word_rank_alignment
from numpy.lib.shape_base import split
```

```
from sklearn import preprocessing
from sklearn.feature_extraction.text import CountVectorizer
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from keras.layers import LSTM,Dense,Dropout,Input,Embedding,Activation,Flatten
from keras.models import Model
import nltk
```

# 3. Read dataset and do preprocessing

In [ ]:
```
data = pd.read_csv(dataset_location,encoding = "ISO-8859-1")
```

In [ ]:
```
data.drop(["Unnamed: 2","Unnamed: 3","Unnamed: 4"],axis = 1,inplace = True)
data.head()
```

Out[ ]:

| | v1 | v2 |
|---|---|---|
| 0 | ham | Go until jurong point, crazy.. Available only ... |
| 1 | ham | Ok lar... Joking wif u oni... |
| 2 | spam | Free entry in 2 a wkly comp to win FA Cup fina... |
| 3 | ham | 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... |

In [ ]:
```
nltk.download('stopwords',quiet=True)
nltk.download('all',quiet=True)
```

Out[ ]:
```
True
```

In [ ]:
```
ps = PorterStemmer()
input = []
```

In [ ]:
```
for i in range(0,5572):
  v2 = data['v2'][i]

  #removing punctuation
  v2 = re.sub('[^a-zA-Z]',' ',v2)

  #converting to lower case
  v2 = v2.lower()

  #splitting the sentence
  v2 = v2.split()

  #removing the stopwords and stemming
  v2 = [ps.stem(word) for word in v2 if not word in set(stopwords.words('english'))]

  v2 = ' '.join(v2)

  input.append(v2)
```

In [ ]:
```
#creating document term matrix
cv = CountVectorizer(max_features=2000)
x = cv.fit_transform(input).toarray()
x.shape
```

Out[ ]: (5572, 2000)

In [ ]:
```
le = preprocessing.LabelEncoder()

data['v1'] = le.fit_transform(data['v1'])
data['v1'].unique()
```

Out[ ]: array([0, 1])

In [ ]:
```
y = data['v1'].values
```

In [ ]:
```
y = y.reshape(-1,1)
```

In [ ]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.4)
```

# 4. Model building - Adding layers, Compiling model and saving model

In [ ]:
```
model = Sequential()
```

In [ ]:
```
model.add(Dense(1565,activation = "relu"))
model.add(Dense(3000,activation = "relu"))
model.add(Dense(1,activation = "sigmoid"))
model.add(Flatten())
```

In [ ]:
```
model.compile(optimizer = "adam",loss = "binary_crossentropy", metrics = ["accuracy
```

In [ ]:
```
model.fit(x_train,y_train,epochs = 15)
```

```
Epoch 1/15
105/105 [==============================] - 13s 93ms/step - loss: 0.1228 - accurac
y: 0.9605
Epoch 2/15
105/105 [==============================] - 7s 68ms/step - loss: 0.0102 - accuracy:
0.9979
Epoch 3/15
105/105 [==============================] - 7s 68ms/step - loss: 0.0031 - accuracy:
0.9991
Epoch 4/15
105/105 [==============================] - 7s 68ms/step - loss: 0.0021 - accuracy:
0.9997
Epoch 5/15
105/105 [==============================] - 7s 67ms/step - loss: 0.0018 - accuracy:
0.9997
Epoch 6/15
105/105 [==============================] - 7s 66ms/step - loss: 0.0018 - accuracy:
0.9997
Epoch 7/15
105/105 [==============================] - 7s 68ms/step - loss: 0.0018 - accuracy:
0.9997
Epoch 8/15
105/105 [==============================] - 7s 69ms/step - loss: 0.0017 - accuracy:
0.9997
Epoch 9/15
105/105 [==============================] - 7s 70ms/step - loss: 0.0017 - accuracy:
0.9997
Epoch 10/15
105/105 [==============================] - 7s 67ms/step - loss: 0.0016 - accuracy:
0.9997
Epoch 11/15
105/105 [==============================] - 7s 67ms/step - loss: 0.0015 - accuracy:
0.9997
Epoch 12/15
105/105 [==============================] - 8s 74ms/step - loss: 0.0017 - accuracy:
0.9997
Epoch 13/15
105/105 [==============================] - 7s 67ms/step - loss: 0.0017 - accuracy:
0.9997
Epoch 14/15
105/105 [==============================] - 7s 67ms/step - loss: 0.0017 - accuracy:
0.9997
Epoch 15/15
105/105 [==============================] - 7s 67ms/step - loss: 0.0017 - accuracy:
0.9997
```

Out[ ]:
```
<keras.callbacks.History at 0x7f793b9e3410>
```

In [ ]:
```
model.save("spam-message-classifier.h5")
```

# 5. Testing the model

In [ ]:
```
ham = "im donee. come pick me up"
spam = "WINNER$$$$ SMS REPLY 'WIN'"
message = re.sub('[^a-zA-Z]',' ',spam)
message
```

Out[ ]:
```
'WINNER     SMS REPLY  WIN '
```

# Testing with spam message

In [ ]:
```
message = message.split()
message = [ps.stem(word) for word in message if not word in set(stopwords.words('er
message = ' '.join(message)
```

In [ ]:
```
message1 = cv.transform([message])
message1
```

Out[ ]:
```
<1x2000 sparse matrix of type '<class 'numpy.int64'>'
        with 4 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
TruePredction = model.predict(message1.astype(float))
```

```
1/1 [==============================] - 0s 149ms/step
```

In [ ]:
```
TruePredction > 0.5
```

Out[ ]:
```
array([[ True]])
```

# Testing with normal message

In [ ]:
```
msg = re.sub('[^a-zA-Z]',' ',ham)
msg
```

Out[ ]:
```
'im donee  come pick me up'
```

In [ ]:
```
msg = msg.split()
msg = [ps.stem(word) for word in msg if not word in set(stopwords.words('english'))
msg = ' '.join(msg)
```

In [ ]:
```
msg
```

Out[ ]:
```
'im done come pick'
```

In [ ]:
```
cv.transform([msg])
```

Out[ ]:
```
<1x2000 sparse matrix of type '<class 'numpy.int64'>'
        with 4 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
FalsePredection = model.predict(cv.transform([msg]))
```

```
1/1 [==============================] - 0s 128ms/step
```

In [ ]:
```
FalsePredection > 0.5
```

Out[ ]:
```
array([[False]])
```