Assignment-4

Problem Statement :- SMS SPAM Classification

|  |  |
| --- | --- |
| Assignment Date | 15 November |
| Student Name | S.Abirami |
| Student Roll Number | 822019104001 |
| Maximum Mark | 2 Mark |

**1. Import required library** In [1]:

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**import** tensorflow

**import** nltk

**from** nltk.corpus **import** stopwords

**from** nltk.stem.porter **import** PorterStemmer

**import** string

**from** tensorflow.keras.preprocessing **import** sequence

**from** keras.models **import** Model, Sequential

**from** keras.preprocessing.text **import** Tokenizer

**from** keras.optimizers **import** Adam, RMSprop

**from** keras.layers **import** Input, Embedding, LSTM, Dense, Flatten, Dropout

**from** sklearn.preprocessing **import** LabelEncoder

**from** sklearn.model\_selection **import** train\_test\_split

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

**Read Dataset** In [2]:

df **=** pd**.**read\_csv(r"C:\Users\manok\Documents\Sem\_7\HX5001-HX6001\Assignment\Assignment\_4\spam.csv", encoding**=**'latin-1') In [3]:

df**.**head()

Out[3]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **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 |

In [4]:

df**.**shape

Out[4]:

(5572, 5)

**Drop Unwanted Column**

In [5]:

df **=** df**.**drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis**=**1)

df **=** df**.**rename(columns**=**{"v2" : "Text", "v1":"Label"})

In [6]:

df**.**head()

Out[6]:

|  |  |  |
| --- | --- | --- |
|  | **Label** | **Text** |
| **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... |

**Remove Duplicate and Null Data**

In [7]:

df**.**isnull()**.**sum()

Out[7]:

Label 0

Text 0

dtype: int64

Label 0

Text 0

dtype: int64

In [8]:

df**.**duplicated()**.**sum()

Out[8]:

403

In [9]:

df **=** df**.**drop\_duplicates(keep**=**'first')

df**.**duplicated()**.**sum()

Out[9]:

0

In [10]:

df**.**shape

Out[10]:

(5169, 2)

**Normalizing the case, Removing the unwanted punctuations, Remove Stopwords**

In [11]:

ps **=** PorterStemmer()

In [12]:

**def** transform\_text(text):

text **=** text**.**lower()

text **=** nltk**.**word\_tokenize(text)

y **=** []

**for** i **in** text:

**if** i**.**isalnum():

y**.**append(i)

text **=** y[:]

y**.**clear()

**for** i **in** text:

**if** i **not** **in** stopwords**.**words('english') **and** i **not** **in** string**.**punctuation:

y**.**append(i)

text **=** y[:]

y**.**clear()

**for** i **in** text:

y**.**append(ps**.**stem(i))

**return** " "**.**join(y)

In[13]:

df['Transformed\_Text'] **=** df['Text']**.**apply(transform\_text)

In [14]:

*# df['Label\_enc'] = df['Label'].map({'ham':0,'spam':1})*

In [15]:

df**.**head()

Out[15]:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Label** | **Text** | **Transformed\_Text** |
| **0** | ham | Go until jurong point, crazy.. Available only ... | go jurong point crazi avail bugi n great world... |
| **1** | ham | Ok lar... Joking wif u oni... | ok lar joke wif u oni |
| **2** | spam | Free entry in 2 a wkly comp to win FA Cup fina... | free entri 2 wkli comp win fa cup final tkt 21... |
| **3** | ham | U dun say so early hor... U c already then say... | u dun say earli hor u c alreadi say |
| **4** | ham | Nah I don't think he goes to usf, he lives aro... | nah think goe usf live around though |

**Counting Words**

In [16]:

avg\_words\_len**=**round(sum([len(i**.**split()) **for** i **in** df['Text']])**/**len(df['Text']))

print(avg\_words\_len)

*# avg\_words\_len=200*

15

In [17]:

s **=** set()

**for** sent **in** df['Transformed\_Text']:

**for** word **in** sent**.**split():

s**.**add(word)

total\_words\_length**=**len(s)

print(total\_words\_length)

*# total\_words\_length=2000*

6736

**3. Create Model**

In [18]:

x **=** df**.**Transformed\_Text

y **=** df**.**Label

le **=** LabelEncoder()

y **=** le**.**fit\_transform(y)

y **=** y**.**reshape(**-**1,1)

In [19]:

*# y = df['Label'].values*

In[20]:

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(x, y, test\_size**=**0.18, random\_state**=**10)

x\_train**.**shape, y\_train**.**shape, x\_test**.**shape, y\_test**.**shape

Out[20]:

((4238,), (4238, 1), (931,), (931, 1))

In [21]:

model **=** Sequential()

**4. Add Layers**

In [22]:

tokenizer **=** Tokenizer(num\_words **=** total\_words\_length, lower **=** **True**)

tokenizer**.**fit\_on\_texts(x\_train)

sequences **=** tokenizer**.**texts\_to\_sequences(x\_train)

x\_train **=** sequence**.**pad\_sequences(sequences, maxlen **=** avg\_words\_len)

**Input Layer**

In [23]:

*# model.add(Input(shape=(1), dtype=tf.string))*

*# model.add(Input(name='inputs',shape=[avg\_words\_len]))*

In [24]:

model**.**add(Embedding(total\_words\_length, 50, input\_length **=** avg\_words\_len))

**LSTM Layer**

In [25]:

model**.**add(LSTM(64))

**Hidden Layer**

In [26]:

model**.**add(Dense(64, activation **=** "relu"))

In [27]:

model**.**add(Flatten())

In [28]:

model**.**add(Dropout(0.2))

In [29]:

model**.**add(Dense(32, activation **=** "relu"))

**Output Layer**

In [30]:

model**.**add(Dense(1, activation **=** 'sigmoid'))

**Model Summary**

In [31]:

model**.**summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, 15, 50) 336800

lstm (LSTM) (None, 64) 29440

dense (Dense) (None, 64) 4160

flatten (Flatten) (None, 64) 0

dropout (Dropout) (None, 64) 0

dense\_1 (Dense) (None, 32) 2080

dense\_2 (Dense) (None, 1) 33

=================================================================

Total params: 372,513

Trainable params: 372,513

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**5. Compile the Model** In [32]:

*# model.compile(loss = 'binary\_crossentropy', optimizer = RMSprop(), metrics = ['accuracy'])*

In [33]:

adam **=** Adam(learning\_rate **=** 0.001, beta\_1 **=** 0.85, beta\_2 **=** 0.97, epsilon **=** 1e-07)

model**.**compile(loss **=** "binary\_crossentropy", optimizer **=** adam, metrics **=** ["accuracy"])

**6. Fit the Model**

In [34]:

epochs**=**5

history **=** model**.**fit(x\_train, y\_train, epochs **=** epochs, validation\_steps**=**0.18, batch\_size**=**10)

Epoch 1/5

424/424 [==============================] - 16s 14ms/step - loss: 0.1346 - accuracy: 0.9552

Epoch 2/5

424/424 [==============================] - 6s 15ms/step - loss: 0.0356 - accuracy: 0.9887

Epoch 3/5

424/424 [==============================] - 6s 15ms/step - loss: 0.0203 - accuracy: 0.9941

Epoch 4/5

424/424 [==============================] - 6s 14ms/step - loss: 0.0096 - accuracy: 0.9969

Epoch 5/5

424/424 [==============================] - 6s 15ms/step - loss: 0.0043 - accuracy: 0.9988

**7. Save the Model**

In [35]:

model**.**save("spam\_analysis.h5")

**8. Test the Model**

In [36]:

test\_sequences **=** tokenizer**.**texts\_to\_sequences(x\_test)

x\_test **=** sequence**.**pad\_sequences(test\_sequences, maxlen**=**avg\_words\_len)

In [37]:

accuracy **=** model**.**evaluate(x\_test, y\_test)

30/30 [==============================] - 2s 10ms/step - loss: 0.2072 - accuracy: 0.9731

In [38]:

**def** predict(message):

txt **=** tokenizer**.**texts\_to\_sequences(message)

txt **=** sequence**.**pad\_sequences(txt, maxlen**=**avg\_words\_len)

pred **=** model**.**predict(txt)

**if** pred**>**0.5:

print("spam")

**else**:

print("Harm")

In [39]:

review1 **=** ["think he goes"]

predict(review1)

1/1 [==============================] - 1s 1s/step

Harm

In[40]:

review2 **=** ["Go until jurong point"]

predict(review2)

1/1 [==============================] - 0s 46ms/step

Harm

In [41]:

review3 **=** ["WINNER!! As a valued network"]

predict(review3)

1/1 [==============================] - 0s 47ms/step

spam

In [42]:

review4 **=** ["URGENT! You have won a 1 week FREE membership"]

predict(review4)

1/1 [==============================] - 0s 44ms/step

spam