

ASSIGNMENT - 4









Problem Statement :- SMS SPAM Classification

| | |
|---------------------------|------------------------|
| Assignment Date | 26 October 2022 |
| Student Name | S.SATHIYA |
| Student Reg Number | 420619104032 |
| Maximum Marks | 2 Marks |

1. Download the Data set:- Data set

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

The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.

|  v1  |  v2  |   |   |
|--|--|---|---|
| class | sms | | |
| ham 87% | 5169 unique values | [null] 99% | [null] 100% |
| spam 13% | | bt not his girlfrnd... 0% | MK17 92H. 450Ppw... 0% |
| | | Other (47) 1% | Other (10) 0% |

[illegible]

2. Import required library

Import the necessary libraries

```
[1]: 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
```

3. Read dataset and do pre-processing

```
In [2]: df = pd.read_csv("spam.csv", encoding='latin-1')
df.head()
```

Out[2]:

| | v1 | v2 | Unnamed: 2 | Unnamed: 3 | Unnamed: 4 |
|---|------|--|------------|------------|------------|
| 0 | ham | Go until jurong point, crazy.. Available only in ... | 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 |

The Columns 2,3,4 will be dropped as they contain no relevant information

```
In [3]: data = df.copy()
data.drop(columns=["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], inplace=True)
data = data.rename(columns={"v1": "label", "v2": "text"})
```

```
In [4]: data.head()
```

Out[4]:

| | label | text |
|---|-------|--|
| 0 | ham | Go until jurong point, crazy.. Available only in ... |
| 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... |

Preproceesing:

```
In [17]: from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Embedding
from tensorflow.keras.callbacks import EarlyStopping

In [18]: # prepare tokenizer
t = Tokenizer()
t.fit_on_texts(X_train)

# integer encode the documents
encoded_train = t.texts_to_sequences(X_train)
encoded_test = t.texts_to_sequences(X_test)

vocab_size = len(t.word_index) + 1

print(encoded_train[0:2])

[[38, 30, 8, 5, 273, 1989, 81, 116, 26, 11, 1656, 322, 10, 53, 18, 299, 30, 349, 1990], [799, 15, 2555, 1442, 1127, 192, 2556, 171, 12, 98, 1991, 44, 195, 1657, 2557, 1992, 2558, 21, 9, 4, 203, 1025, 225]]
```

4. Create Model

```
In [7]: data['label'] = data['label'].map( {'spam': 1, 'ham': 0} )
data.head()
```

```
Out[7]:
```

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

```
In [8]: data_ham = data[data['label'] == 0].copy()
data_spam = data[data['label'] == 1].copy()
```

WordClouds

WordClouds

```
In [9]: def show_wordcloud(df, title):
text = ' '.join(df['text']).astype(str).tolist()
stopwords = set(wordcloud.STOPWORDS)

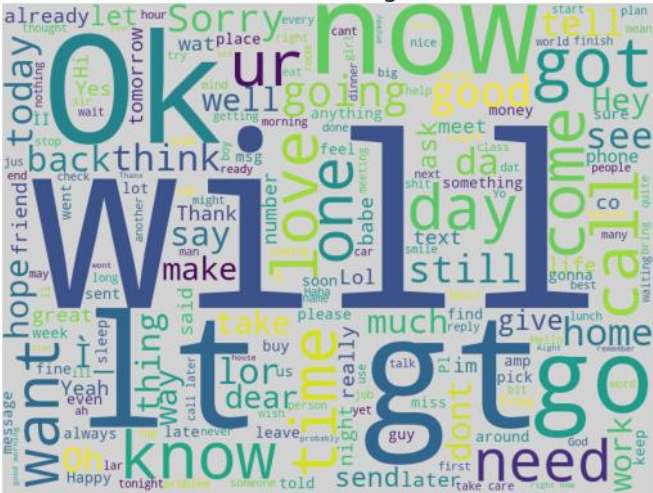
fig_wordcloud = wordcloud.WordCloud(stopwords=stopwords, background_color='lightgrey',
colormap='viridis', width=800, height=600).generate(text)

plt.figure(figsize=(10,7), frameon=True)
plt.imshow(fig_wordcloud)
plt.axis('off')
plt.title(title, fontsize=20)
plt.savefig("wd.png")
plt.show()
```

WordCloud: Ham messages

```
In [10]: show_wordcloud(data_ham, "Ham messages")
```

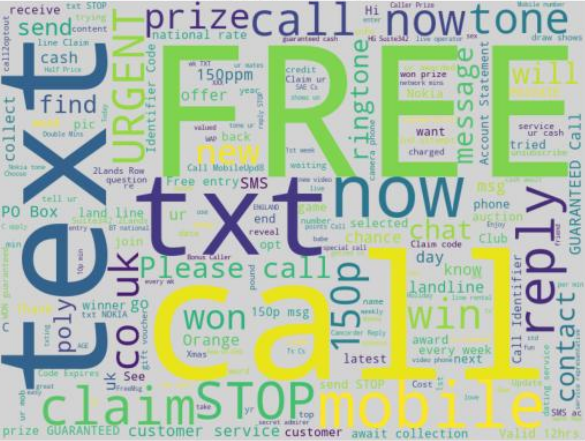
Ham messages



WordCloud: Spam messages

```
In [11]: show_wordcloud(data_spam, "Spam messages")
```

Spam messages



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

6. Compile the Model

```
In [19]: # pad documents to a max length of 4 words
max_length = 8
padded_train = pad_sequences(encoded_train, maxlen=max_length, padding='post')
padded_test = pad_sequences(encoded_test, maxlen=max_length, padding='post')

print(padded_train)
```

```
[[ 322  10  53 ...  30 349 1990]
 [1992 2558  21 ... 203 1025 225]
 [  83 1443   4 ...   2 3794 3795]
 ...
 [1477  30 2063 ... 239  30 2064]
 [ 763 1679 1161 ...   0   0   0]
 [   8  155  20 ...   8 290 175]]
```

```
In [20]: # define the model
model = Sequential()
model.add(Embedding(vocab_size, 24, input_length=max_length))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# compile the model
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])

# summarize the model
print(model.summary())
```

Model: "sequential_1"

| Layer (type) | Output Shape | Param # |
|---------------------------|---------------|---------|
| ===== | | |
| embedding_1 (Embedding) | (None, 8, 24) | 190920 |
| flatten_1 (Flatten) | (None, 192) | 0 |
| dense_2 (Dense) | (None, 500) | 96500 |
| dense_3 (Dense) | (None, 200) | 100200 |
| dropout (Dropout) | (None, 200) | 0 |
| dense_4 (Dense) | (None, 100) | 20100 |
| dense_5 (Dense) | (None, 1) | 101 |
| ===== | | |
| Total params: 407,821 | | |
| Trainable params: 407,821 | | |
| Non-trainable params: 0 | | |
| ===== | | |
| None | | |

7. Fit the Model

```
In [21]: early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)

# fit the model
model.fit(x=padded_train,
          y=y_train,
          epochs=50,
          validation_data=(padded_test, y_test), verbose=1,
          callbacks=[early_stop])
```

Epoch 1/50
140/140 [=====] - 1s 4ms/step - loss: 0.2034 - accuracy: 0.9195 - val_loss: 0.1061 - val_accuracy: 0.9758
Epoch 2/50
140/140 [=====] - 0s 3ms/step - loss: 0.0447 - accuracy: 0.9865 - val_loss: 0.0840 - val_accuracy: 0.9821
Epoch 3/50
140/140 [=====] - 0s 3ms/step - loss: 0.0136 - accuracy: 0.9969 - val_loss: 0.0997 - val_accuracy: 0.9839
Epoch 4/50
140/140 [=====] - 0s 3ms/step - loss: 6.0631e-04 - accuracy: 0.9998 - val_loss: 0.2119 - val_accuracy: 0.9830
Epoch 5/50
140/140 [=====] - 0s 3ms/step - loss: 1.2411e-06 - accuracy: 1.0000 - val_loss: 0.2899 - val_accuracy: 0.9803
Epoch 6/50
140/140 [=====] - 0s 3ms/step - loss: 3.1918e-08 - accuracy: 1.0000 - val_loss: 0.2903 - val_accuracy: 0.9821
Epoch 7/50
140/140 [=====] - 0s 3ms/step - loss: 4.8863e-09 - accuracy: 1.0000 - val_loss: 0.2921 - val_accuracy: 0.9830
Epoch 8/50
140/140 [=====] - 0s 2ms/step - loss: 9.7544e-10 - accuracy: 1.0000 - val_loss: 0.2946 - val_accuracy: 0.9830
Epoch 9/50
140/140 [=====] - 0s 3ms/step - loss: 1.3770e-09 - accuracy: 1.0000 - val_loss: 0.3048 - val_accuracy: 0.9821
Epoch 10/50
140/140 [=====] - 0s 3ms/step - loss: 1.3219e-09 - accuracy: 1.0000 - val_loss: 0.3032 - val_accuracy: 0.9812
Epoch 11/50
140/140 [=====] - 0s 3ms/step - loss: 1.1548e-09 - accuracy: 1.0000 - val_loss: 0.3015 - val_accuracy: 0.9830
Epoch 12/50
140/140 [=====] - 0s 3ms/step - loss: 8.7392e-10 - accuracy: 1.0000 - val_loss: 0.3087 - val_accuracy: 0.9830

8. Save The Model

```
In [29]: model.save("spam_model")
```

WARNING:tensorflow:From /Users/mac/opt/anaconda3/envs/deeplearning/lib/python3.7/site-packages/tensorflow/python/training/tracking/tracking.py:111: Model.state_updates (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
WARNING:tensorflow:From /Users/mac/opt/anaconda3/envs/deeplearning/lib/python3.7/site-packages/tensorflow/python/training/tracking/tracking.py:111: Layer.updates (from tensorflow.python.keras.engine.base_layer) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
INFO:tensorflow:Assets written to: spam_model/assets

```
In [30]: with open('spam_model/tokenizer.pkl', 'wb') as output:
          pickle.dump(t, output, pickle.HIGHEST_PROTOCOL)
```

9. Test The Model

```
In [31]: s_model = tf.keras.models.load_model("spam_model")

with open('spam_model/tokenizer.pkl', 'rb') as input:
    tokenizer = pickle.load(input)

# s_model.summary()
```

```
In [38]: sms_spam = ["We know someone who you know that fancies you. Call 09058097218 to find out who. POBox 6, LS15HB "]
sms_ham = ["I'll text Tanya when I get home, hang on"]

sms_proc = tokenizer.texts_to_sequences(sms_ham)
sms_proc = pad_sequences(sms_proc, maxlen=max_length, padding='post')

pred = (model.predict(sms_proc) > 0.5).astype("int32").item()
pred
```

```
In [39]: pred = (model.predict(sms_proc) > 0.5).astype("int32").item()
pred
```

```
Out[39]: 0
```

```
In [33]: X_test[5]
```

```
Out[33]: "I'll text carlos and let you know, hang on"
```

```
True
```