In [1]:

```
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import time
```

In [2]:

```
df = pd.read_csv(r'C:\Users\jgaur\Tensorflow_Tut\NLP\train.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

In [4]:

```
df.shape
```

Out[4]:

```
(7613, 5)
```

In [5]:

```
print((df.target == 1).sum()) # disaster
print((df.target == 0).sum()) # no disaster
```

```
3271
4342
```

In [6]:

```
# Preprocessing

import re # Regular Expression
import string

    def remove_url(text):
        url = re.compile(r"https?://\S+|www\.\S+")
        return url.sub(r"", text)

def remove_punc(text):
    translator = str.maketrans("", "", string.punctuation)
    return text.translate(translator)

string.punctuation
```

Out[6]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [7]:

```
pattern = re.compile(r"https?://(\S+|www)\.\S+")
for t in df.text:
    matches = pattern.findall(t)
    for match in matches:
        print("1")
        print(t)
        print("2")
        print(match)
        print("3")
        print(pattern.sub(r"", t))
    if len(matches) > 0:
        break
```

```
1
@bbcmtd Wholesale Markets ablaze http://t.co/lHYXEOHY6C (http://t.co/lHYXEOHY6C)
2
t
3
@bbcmtd Wholesale Markets ablaze
```

In [8]:

```
df['text'] = df.text.map(remove_url)        # map(lambda x: remove_url(x))
df['text'] = df.text.map(remove_punc)
df.head()
```

Out[8]:

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this earthquake Ma...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask Canada	1
2	5	NaN	NaN	All residents asked to shelter in place are be...	1
3	6	NaN	NaN	13000 people receive wildfires evacuation orde...	1
4	7	NaN	NaN	Just got sent this photo from Ruby Alaska as s...	1

In [9]:

```
# remove stopwords
# (Stopwords are the English words which does not add much meaning to a sentence. They can safely be
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

stop = set(stopwords.words("english"))

def remove_stopwords(text):
    filtered_words = [word.lower() for word in text.split() if word.lower() not in stop]
    return " ".join(filtered_words)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\jgaur\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [10]:

```
df["text"] = df.text.map(remove_stopwords)
df.text
```

Out[10]:

```
0            deeds reason earthquake may allah forgive us
1                   forest fire near la ronge sask canada
2       residents asked shelter place notified officer...
3       13000 people receive wildfires evacuation orde...
4       got sent photo ruby alaska smoke wildfires pou...
                              ...
7608    two giant cranes holding bridge collapse nearb...
7609    ariaahrary thetawniest control wild fires cali...
7610                      m194 0104 utc5km volcano hawaii
7611    police investigating ebike collided car little...
7612    latest homes razed northern california wildfir...
Name: text, Length: 7613, dtype: object
```

In [11]:

```python
from collections import Counter

# Count unique words
def counter_word(text_col):
    count = Counter()
    for text in text_col.values:
        for word in text.split():
            count[word] += 1
    return count

counter = counter_word(df.text)
```

In [12]:

```python
len(counter)
```

Out[12]:

```
17971
```

In [13]:

```python
# counter
```

In [14]:

```python
counter.most_common(5)
```

Out[14]:

```
[('like', 345), ('im', 299), ('amp', 298), ('fire', 250), ('get', 229)]
```

In [15]:

```python
num_unique_words = len(counter)
```

In [16]:

```python
train_size = int(df.shape[0] * 0.8)
train_df = df[:train_size]
val_df = df[train_size:]

train_sentence = train_df.text.to_numpy()
train_labels = train_df.target.to_numpy()
val_sentence = val_df.text.to_numpy()
val_labels = val_df.target.to_numpy()
```

In [17]:

```python
train_sentence.shape, val_sentence.shape
train_sentence
```

Out[17]:

```
array(['deeds reason earthquake may allah forgive us',
       'forest fire near la ronge sask canada',
       'residents asked shelter place notified officers evacuation shelter place orders expected',
       ..., 'feel like sinking unhappiness take quiz',
       'sinking music video tv career brooke hogan thanking dad free publicityalthough doubt help',
       'supernovalester feel bad literally feel feeling heart sinking bc didnt get anyone ugh jf
c'],
      dtype=object)
```

In [18]:

```python
# tokenize

from tensorflow.keras.preprocessing.text import Tokenizer

# vectorize a text corpus by turning each text into a sequence of interger
tokenizer = Tokenizer(num_words=num_unique_words)
tokenizer.fit_on_texts(train_sentence)  # fit only to training
```

In [19]:

```python
# each word has unique index
word_index = tokenizer.word_index
```

In [20]:

```python
# word_index
```

In [21]:
```
train_sequence = tokenizer.texts_to_sequences(train_sentence)
val_sequence = tokenizer.texts_to_sequences(val_sentence)
```

In [22]:
```
print(train_sentence[0:5])
print(train_sequence[0:5])
```

```
['deeds reason earthquake may allah forgive us'
 'forest fire near la ronge sask canada'
 'residents asked shelter place notified officers evacuation shelter place orders expected'
 '13000 people receive wildfires evacuation orders california'
 'got sent photo ruby alaska smoke wildfires pours school']
[[3739, 696, 235, 41, 1282, 3740, 14], [71, 3, 129, 576, 5670, 5671, 1283], [1448, 1186, 1882, 495,
5672, 1449, 116, 1882, 495, 976, 1187], [2243, 8, 3741, 1070, 116, 976, 24], [27, 1071, 358, 5673,
1635, 892, 1070, 5674, 91]]
```

In [23]:
```
# pad the sentence to have the same length
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Maximum number of words in a suquence
max_length = 20

train_padded = pad_sequences(train_sequence, maxlen=max_length, padding="post", truncating="post")
val_padded = pad_sequences(val_sequence, maxlen=max_length, padding="post", truncating="post")
train_padded.shape, val_padded.shape
```

Out[23]:
```
((6090, 20), (1523, 20))
```

In [24]:
```
train_padded[0]
```

Out[24]:
```
array([3739,  696,  235,   41, 1282, 3740,   14,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0])
```

In [25]:
```
print(train_sentence[10])
print(train_sequence[10])
print(train_padded[10])
```

```
three people died heat wave far
[520, 8, 395, 156, 297, 411]
[520   8 395 156 297 411   0   0   0   0   0   0   0   0   0   0   0   0
   0   0]
```

In [26]:
```
# Check reversing the indices

# flip key, values
reverse_word_index = dict([(idx, word) for word, idx in word_index.items()])
```

In [27]:
```
# reverse_word_index
```

In [28]:
```
def decode(sequence):
    return ' '.join([reverse_word_index.get(idx, "?") for idx in sequence])
```

In [29]:
```
decode_text = decode(train_sequence[10])
print(train_sequence[10])
print(decode_text)
```

```
[520, 8, 395, 156, 297, 411]
three people died heat wave far
```

In [30]:

```python
# Create LSTM Model
from tensorflow.keras import layers

# Embedding: https://www.tensorflow.org/tutorials/text/word_embeddings
# Turns positive integers (indexes) into dense vectors of fixed size. (other approach could be one-h

model = keras.models.Sequential()
model.add(layers.Embedding(num_unique_words, 32, input_length=max_length))

# The layer will take as input an integer matrix of size (batch, input_length)
# and  the largest integer (i.e. word index) in the input should be no longer than num_words (vocabu
# Now model.output_shape is (None, input_length, 32), where None is tha batch dimension

model.add(layers.LSTM(64, dropout=0.1))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 20, 32)            575072    
_________________________________________________________________
lstm (LSTM)                  (None, 64)                24832     
_________________________________________________________________
dense (Dense)                (None, 1)                 65        
=================================================================
Total params: 599,969
Trainable params: 599,969
Non-trainable params: 0
_________________________________________________________________
```

In [31]:

```python
loss = keras.losses.BinaryCrossentropy(from_logits=False)
optim = keras.optimizers.Adam(lr=0.001)
metrics = ['accuracy']

model.compile(loss=loss, optimizer=optim, metrics=metrics)
```

In [32]:
```
model.fit(train_padded, train_labels, epochs=20, validation_data=(val_padded, val_labels), verbose=2
```

```
Epoch 1/20
191/191 - 6s - loss: 0.5350 - accuracy: 0.7241 - val_loss: 0.4956 - val_accuracy: 0.7630
Epoch 2/20
191/191 - 6s - loss: 0.2864 - accuracy: 0.8903 - val_loss: 0.5628 - val_accuracy: 0.7663
Epoch 3/20
191/191 - 5s - loss: 0.1507 - accuracy: 0.9489 - val_loss: 0.6378 - val_accuracy: 0.7498
Epoch 4/20
191/191 - 5s - loss: 0.1039 - accuracy: 0.9673 - val_loss: 0.6368 - val_accuracy: 0.7479
Epoch 5/20
191/191 - 6s - loss: 0.0864 - accuracy: 0.9732 - val_loss: 0.7689 - val_accuracy: 0.7334
Epoch 6/20
191/191 - 6s - loss: 0.0732 - accuracy: 0.9772 - val_loss: 0.8256 - val_accuracy: 0.7321
Epoch 7/20
191/191 - 7s - loss: 0.0649 - accuracy: 0.9777 - val_loss: 0.9546 - val_accuracy: 0.7255
Epoch 8/20
191/191 - 7s - loss: 0.0534 - accuracy: 0.9782 - val_loss: 1.1011 - val_accuracy: 0.7393
Epoch 9/20
191/191 - 9s - loss: 0.0477 - accuracy: 0.9796 - val_loss: 0.8947 - val_accuracy: 0.7288
Epoch 10/20
191/191 - 8s - loss: 0.0410 - accuracy: 0.9819 - val_loss: 1.1788 - val_accuracy: 0.7387
Epoch 11/20
191/191 - 7s - loss: 0.0363 - accuracy: 0.9831 - val_loss: 1.3008 - val_accuracy: 0.7446
Epoch 12/20
191/191 - 7s - loss: 0.0387 - accuracy: 0.9828 - val_loss: 1.2571 - val_accuracy: 0.7203
Epoch 13/20
191/191 - 5s - loss: 0.0357 - accuracy: 0.9823 - val_loss: 1.5327 - val_accuracy: 0.7249
Epoch 14/20
191/191 - 6s - loss: 0.0338 - accuracy: 0.9841 - val_loss: 1.8678 - val_accuracy: 0.7216
Epoch 15/20
191/191 - 6s - loss: 0.0422 - accuracy: 0.9816 - val_loss: 1.3540 - val_accuracy: 0.7295
Epoch 16/20
191/191 - 6s - loss: 0.0448 - accuracy: 0.9818 - val_loss: 1.1771 - val_accuracy: 0.7150
Epoch 17/20
191/191 - 6s - loss: 0.0364 - accuracy: 0.9823 - val_loss: 1.1933 - val_accuracy: 0.7400
Epoch 18/20
191/191 - 6s - loss: 0.0320 - accuracy: 0.9837 - val_loss: 1.5888 - val_accuracy: 0.7249
Epoch 19/20
191/191 - 6s - loss: 0.0354 - accuracy: 0.9828 - val_loss: 1.8026 - val_accuracy: 0.7216
Epoch 20/20
191/191 - 6s - loss: 0.0416 - accuracy: 0.9813 - val_loss: 1.3057 - val_accuracy: 0.7229
```

Out[32]:
```
<tensorflow.python.keras.callbacks.History at 0x1e12a269d60>
```

In [33]:
```
predictions = model.predict(train_padded)
predictions = [1 if p> 0.5 else 0 for p in predictions]
```

In [34]:
```
print(train_sentence[10:20])
```

```
['three people died heat wave far'
 'haha south tampa getting flooded hah wait second live south tampa gonna gonna fvck flooding'
 'raining flooding florida tampabay tampa 18 19 days ive lost count'
 'flood bago myanmar arrived bago'
 'damage school bus 80 multi car crash breaking' 'whats man' 'love fruits'
 'summer lovely' 'car fast' 'goooooooaaaaaal']
```

In [35]:
```
print(train_labels[10:20])
print(predictions[10:20])
```

```
[1 1 1 1 1 0 0 0 0 0]
[1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
```

In [ ]:

```
```