In [51]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import feature_extraction
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re

from sklearn.model_selection import train_test_split

from gensim.models import Word2Vec

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding, Bidirectional
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [52]:

```python
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/nlp-getting-started/sample_submission.csv
/kaggle/input/nlp-getting-started/train.csv
/kaggle/input/nlp-getting-started/test.csv
```

Task Description

The objective of this task is to predict whether a tweet is referring to a disaster or non-disaster situation. The data comes from the Kaggle Competition: 'Natural Language Processing with Disaster Tweets' (https://www.kaggle.com/c/nlp-getting-started/overview), and consists of a training data set of 7613 tweets, each labeled as either '1' for being about a disaster, or '0' for not being about a disaster. The training data also includes keyword and location features.

In [53]:

```python
data = pd.read_csv('/kaggle/input/nlp-getting-started/train.csv')
data.head(10)
```

Out[53]:

	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
5	8	NaN	NaN	#RockyFire Update => California Hwy. 20 closed...	1
6	10	NaN	NaN	#flood #disaster Heavy rain causes flash flood...	1
7	13	NaN	NaN	I'm on top of the hill and I can see a fire in...	1
8	14	NaN	NaN	There's an emergency evacuation happening now ...	1
9	15	NaN	NaN	I'm afraid that the tornado is coming to our a...	1

The first 10 entries demomstrate some of the cleaning needed for the text. The text needs to be shifted into all lower case, and additional characters like numbers and symbols need to be removed.

In [54]:
```
# The first five non-disaster tweets
data[data["target"] == 0]["text"].values[0:5]
```

Out[54]:
```
array(["What's up man?", 'I love fruits', 'Summer is lovely',
       'My car is so fast', 'What a goooooooaaaaaal!!!!!!'], dtype=object)
```

In [55]:
```
# The first five disaster tweets
data[data["target"] == 1]["text"].values[0:5]
```

Out[55]:
```
array(['Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all',
       'Forest fire near La Ronge Sask. Canada',
       "All residents asked to 'shelter in place' are being notified by officers. No
other evacuation or shelter in place orders are expected",
       '13,000 people receive #wildfires evacuation orders in California ',
       'Just got sent this photo from Ruby #Alaska as smoke from #wildfires pours int
o a school '],
      dtype=object)
```

In [56]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        7613 non-null   int64
 1   keyword   7552 non-null   object
 2   location  5080 non-null   object
 3   text      7613 non-null   object
 4   target    7613 non-null   int64
dtypes: int64(2), object(3)
memory usage: 297.5+ KB
```

In [57]:
```
data.nunique()
```

Out[57]:
```
id          7613
keyword      221
location    3341
text        7503
target         2
dtype: int64
```

EDA and Data Cleaning

In this data set, the columns of 'keyword' and 'location' will be removed because of th enumber of missing values. Furthermore, 110 duplicated tweets will be removed. There are no missing values in the texts or target label.

Visualizations will illustrate the proportion of tweets labeled as 'disaster', and the distribution ofnumber of words in the disaster and non-disaster tweets. Finally, the tweets will be cleaned, putting all letters in lower case, removing unicode characters, and removing the most common English words.

In [58]:
```
# Dropping 'keyword' and 'location' columns because of the number of null values.

data = data.drop(['keyword', 'location'], axis=1)
data.head()
```

Out[58]:

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

In [59]:

```
# Identifying the texts that are repeated
print(data[data.duplicated(['text'], keep='first')])
print(f"There are {data[data.duplicated(['text'], keep='first')].shape[0]} duplicated
```

```
         id                                               text  target
48       68  Check these out: http://t.co/rOI2NSmEJJ http:/...       0
115     165  320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/vA...       0
119     172  320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/TH...       0
164     238  Experts in France begin examining airplane deb...       1
624     898                       To fight bioterrorism sir.       0
...     ...                                                ...     ...
7600  10855  Evacuation order lifted for town of Roosevelt:...       1
7607  10867  #stormchase Violent Record Breaking EF-5 El Re...       1
7609  10870  @aria_ahrary @TheTawniest The out of control w...       1
7610  10871  M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...       1
7611  10872  Police investigating after an e-bike collided ...       1

[110 rows x 3 columns]
There are 110 duplicated tweets.
```

In [60]:

```
print("Sample duplicated tweets:")
print(data[data.duplicated(['text'], keep=False)].iloc[0, 1])
print(data[data.duplicated(['text'], keep=False)].iloc[1, 1])
print(data[data.duplicated(['text'], keep=False)].iloc[2, 1])
print(data[data.duplicated(['text'], keep=False)].iloc[3, 1])
print(data[data.duplicated(['text'], keep=False)].iloc[4, 1])
```

```
Sample duplicated tweets:
Check these out: http://t.co/rOI2NSmEJJ http://t.co/3Tj8ZjiN21 http://t.co/YDUiXEfIpE
http://t.co/LxTjc87KLS #nsfw
Check these out: http://t.co/rOI2NSmEJJ http://t.co/3Tj8ZjiN21 http://t.co/YDUiXEfIpE
http://t.co/LxTjc87KLS #nsfw
320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/vAM5POdGyw | @djicemoon | #Dubstep #TrapM
usic #DnB #EDM #Dance #Ices  Û_ http://t.co/zEVakJaPcz
320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/vAM5POdGyw | @djicemoon | #Dubstep #TrapM
usic #DnB #EDM #Dance #Ices  Û_ http://t.co/zEVakJaPcz
320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/THyzOMVWU0 | @djicemoon | #Dubstep #TrapM
usic #DnB #EDM #Dance #Ices  Û_ http://t.co/83jOO0xk29
```

In [61]:

```
# Dropping duplicate tweets
data = data.drop_duplicates(['text'])
data.nunique()
```

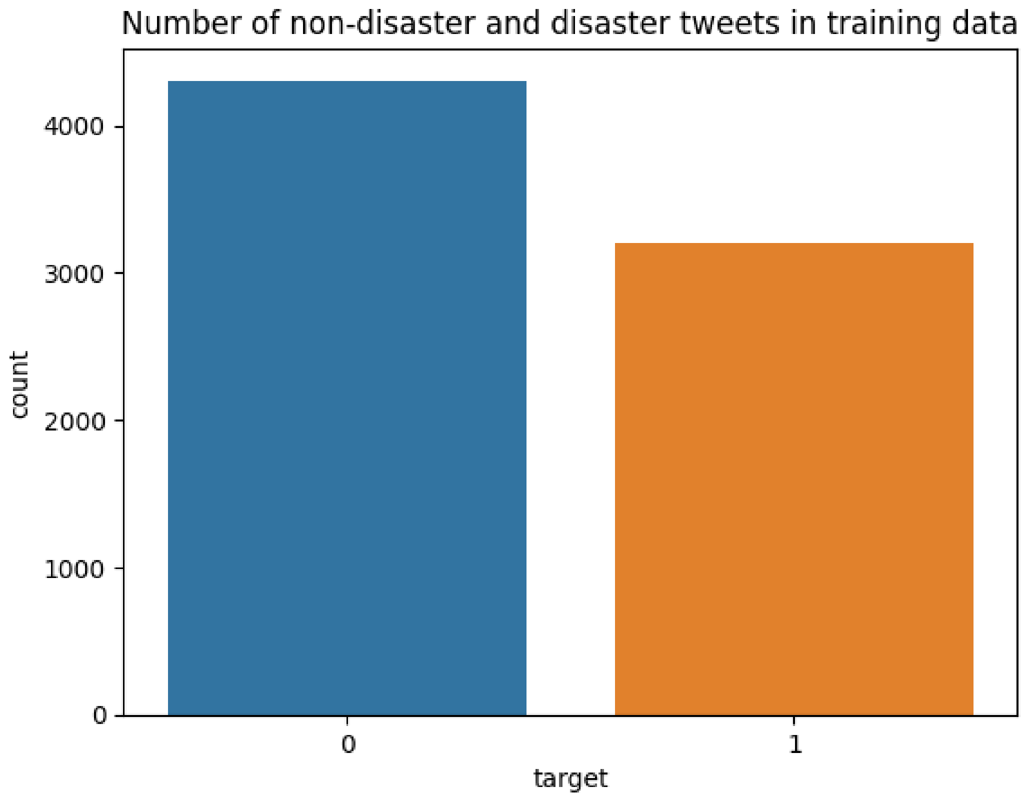
Out[61]:

```
id        7503
text      7503
target       2
dtype: int64
```

In [62]:

```
sns.countplot(data=data, x='target').set(title='Number of non-disaster and disaster tw
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_c
ategorical_dtype is deprecated and will be removed in a future version. Use isinstanc
e(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_c
ategorical_dtype is deprecated and will be removed in a future version. Use isinstanc
e(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_c
ategorical_dtype is deprecated and will be removed in a future version. Use isinstanc
e(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



In [63]:

```
# Calculating the lengths of each tweet
lengths = [len(text.split()) for text in data['text']]
data['lengths'] = lengths
data.head()
```

Out[63]:

	id	text	target	lengths
0	1	Our Deeds are the Reason of this #earthquake M...	1	13
1	4	Forest fire near La Ronge Sask. Canada	1	7
2	5	All residents asked to 'shelter in place' are ...	1	22
3	6	13,000 people receive #wildfires evacuation or...	1	8
4	7	Just got sent this photo from Ruby #Alaska as ...	1	16

In [64]:

```
data.describe()
```

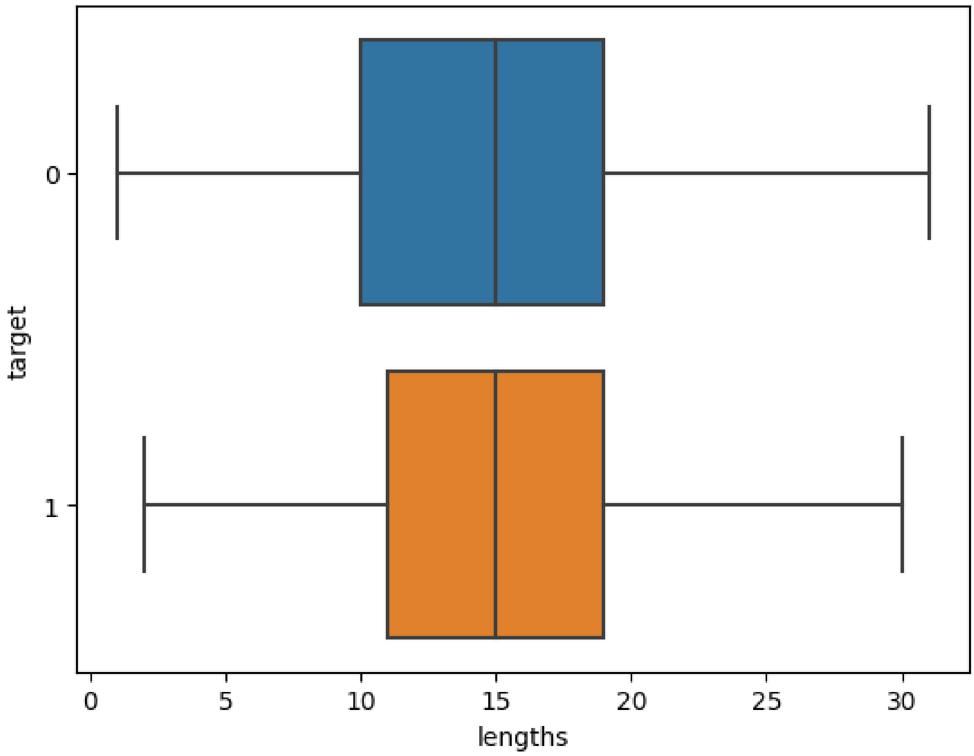
Out[64]:

	id	target	lengths
count	7503.000000	7503.000000	7503.000000
mean	5439.831401	0.426230	14.876849
std	3141.748725	0.494561	5.735043
min	1.000000	0.000000	1.000000
25%	2726.500000	0.000000	11.000000
50%	5408.000000	0.000000	15.000000
75%	8149.500000	1.000000	19.000000
max	10873.000000	1.000000	31.000000

In [65]:

```
sns.boxplot(data=data, x='lengths', y='target', orient='h')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_c
ategorical_dtype is deprecated and will be removed in a future version. Use isinstanc
e(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_c
ategorical_dtype is deprecated and will be removed in a future version. Use isinstanc
e(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_c
ategorical_dtype is deprecated and will be removed in a future version. Use isinstanc
e(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



The appears to be a similar disribution in lengths between non-disaster and disaster tweets. The shortest tweet is one word, and the longest 31 words, which are appropriate lengths for tweets. The median length for both categories is approximately 15 words per tweet.

In [66]:

```
#Data proprocessing: preparing the texts by removing common words, putting all letters

stop = stopwords.words('english')

def tweet_cleaner(tweet):
    """
```

```
    This function converts all letters to lowercase, removes unicode characters and re

    Parameter:
    tweet(str): the text of the tweet

    Returns:
    cleaned tweet.
    """
    tmp = tweet.lower()
    tmp = re.sub(r"(@\[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)|^rt|http.+?", "",
    tmp = " ".join([word for word in tmp.split() if word not in (stop)])

    return tmp
```

In [67]:
```
# Sample text before and after cleaning
tweet = data['text'][5]
print(tweet)
tweet_cleaner(tweet)
```

```
#RockyFire Update => California Hwy. 20 closed in both directions due to Lake County
fire - #CAfire #wildfires
```

Out[67]:
```
'rockyfire update california hwy 20 closed directions due lake county fire cafire wil
dfires'
```

In [68]:
```
# Cleaning the tweets
text_clean = data['text'].apply(tweet_cleaner)
text_clean[0:5]
```

Out[68]:
```
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...
Name: text, dtype: object
```

In [69]:
```
y = data['target']
y
```

Out[69]:
```
0       1
1       1
2       1
3       1
4       1
       ..
7604    1
7605    1
7606    1
7608    1
7612    1
Name: target, Length: 7503, dtype: int64
```

Model Architecture

The modeling will coming in two parts: transforming the text into vectors, and creating LSTM models to predict if the tweet is about a disaster. For the transformation from text to vectors, each cleaned text will be fit on a tokenizer model, and padded with zeros so all texts are the same length. After this process, each word is replaced by a number that corresponds to word, maintaining the same word order as the original text. In contrast, word2vec identifies the nearby words frequently found by each word.

In [70]:
```
# Using the Tokenizer function to count and encode words
tokenizer = Tokenizer()
tokenizer.fit_on_texts(text_clean)

X = tokenizer.texts_to_sequences(text_clean)
vocab_size = len(tokenizer.word_index)+1
```

In [71]:
```python
print(f"The length of the tokenized texts is {len(X)}.")
print(f"The size of the vocabulary is {vocab_size} words.")
```

```
The length of the tokenized texts is 7503.
The size of the vocabulary is 17845 words.
```

In [72]:
```python
# Comparing the tweets before and after cleaning and tokenizing.
#The clean tweets are 'padded' with zeros so each one is the same length, 31 words.

print("Sample tweet before cleaning:\n{}".format(data['text'][6]))
print("Tweet after cleaning:\n{}".format(text_clean[6]))
print("\nAfter tokenizing :\n{}".format(X[6]))

X = pad_sequences(X, padding='post')
print("\nAfter padding :\n{}".format(X[6]))
```

```
Sample tweet before cleaning:
#flood #disaster Heavy rain causes flash flooding of streets in Manitou, Colorado Spr
ings areas
Tweet after cleaning:
flood disaster heavy rain causes flash flooding streets manitou colorado springs area
s

After tokenizing :
[127, 16, 696, 188, 1038, 697, 149, 1551, 6461, 860, 2180, 1419]

After padding :
[ 127   16  696  188 1038  697  149 1551 6461  860 2180 1419    0    0
    0    0    0    0    0    0    0    0    0    0    0]
```

In [73]:
```python
# The maximum length of the cleaned texts is 25.  All the tokenized texts are padded w
# zeros to reach the length of 25.
max_length = X.shape[1]
max_length
```

Out[73]:
```
25
```

In [74]:
```python
# Separating each word in a tweet into an individual token
tokenized_corpus = [word_tokenize(tweets) for tweets in text_clean]
print(f"Sample tweet: {text_clean[6]}")
print(f"Sample tokenized tweet: {tokenized_corpus[6]}")
print(f"Number of texts in tokenized corpus: {len(tokenized_corpus)}")
```

```
Sample tweet: flood disaster heavy rain causes flash flooding streets manitou colorad
o springs areas
Sample tokenized tweet: ['flood', 'disaster', 'heavy', 'rain', 'causes', 'flash', 'fl
ooding', 'streets', 'manitou', 'colorado', 'springs', 'areas']
Number of texts in tokenized corpus: 7503
```

In [75]:
```python
# Initializing the Word2Vec model
word2vec_model = Word2Vec(sentences=tokenized_corpus, vector_size=100, window=5, sg=1,
```

In [76]:
```python
# Example of the most similar words to 'disaster'
similar_words = word2vec_model.wv.most_similar(positive=["disaster"], topn=10)
for word, similarity in similar_words:
    print(f"{word}: {similarity}")
```

```
typhoondevastated: 0.9931120872497559
obama: 0.9912415742874146
saipan: 0.9911453127861023
declares: 0.9866383075714111
signs: 0.9821597933769226
declaration: 0.9688212871551514
marians: 0.9337175488471985
natural: 0.9269368052482605
northern: 0.9064761400222778
abcnews: 0.9016797542572021
```

In [77]:
```python
word2vec_model.wv.index_to_key[0:20]
```

Out[77]:
```
['like',
 'im',
 'amp',
 'fire',
 'get',
 'new',
 'via',
 'dont',
 'people',
 'one',
 'news',
 'us',
 'video',
 '2',
 'emergency',
 'disaster',
 'police',
 'would',
 'still',
 'body']
```

In [78]:
```
for i in range(10):
    print(f" '{word2vec_model.wv.index_to_key[i]}' is similar to {word2vec_model.wv.mo
```

```
 'like' is similar to [('im', 0.9953508377075195), ('going', 0.9952459335327148), ('k
now', 0.9950524568557739), ('think', 0.9946407675743103), ('back', 0.994627237319946
3)].
 'im' is similar to [('like', 0.9953508973121643), ('going', 0.9947434067726135), ('s
ee', 0.9944233894348145), ('know', 0.9943664073944092), ('think', 0.993981897830963
1)].
 'amp' is similar to [('shit', 0.9967986345291138), ('love', 0.9967678189277649), ('m
uch', 0.996654748916626), ('take', 0.9966371059417725), ('u', 0.9966089725494385)].
 'fire' is similar to [('truck', 0.9892762303352356), ('buildings', 0.988390207290649
4), ('township', 0.987760066986084), ('burning', 0.9872680902481079), ('rocky', 0.986
3860011100769)].
 'get' is similar to [('swallowed', 0.9949874877929688), ('airport', 0.99423092603683
47), ('sandstorm', 0.9940783381462097), ('watch', 0.9931297302246094), ('minute', 0.9
918684959411621)].
 'new' is similar to [('goes', 0.995222806930542), ('reddits', 0.9949044585227966),
('effect', 0.9943391680717468), ('policy', 0.9941672682762146), ('many', 0.9935449957
847595)].
 'via' is similar to [('flag', 0.9911137819290161), ('pamela', 0.9907219409942627),
('israeli', 0.9892648458480835), ('geller', 0.9891414046287537), ('waving', 0.9873414
039611816)].
 'dont' is similar to [('know', 0.9943250417709351), ('like', 0.9936843514442444),
('want', 0.9930048584938049), ('im', 0.9929423332214355), ('think', 0.992498099803924
6)].
 'people' is similar to [('60', 0.9950867295265198), ('thats', 0.9945511817932129),
('shots', 0.9945022463798523), ('dead', 0.9944791197776794), ('screams', 0.9944607019
424438)].
 'one' is similar to [('ive', 0.9967653751373291), ('day', 0.996748149394989), ('go
t', 0.9966810345649719), ('another', 0.9965003132820129), ('see', 0.996272027492523
2)].
```

Preparing the training data for LSTM models.

In [79]:
```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [80]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(6002, 25)
(1501, 25)
(6002,)
(1501,)
```

Model Architecture

The base model is a LSTM model with embedding, LSTM and dense layer, before the final classification layer with sigmoid activation. The initial 3 models have units of 32, 64 and 128. They will be compiled with the Adam optimizer, binary crossentropy loss function, and accuracy metric, and trained over 20 epochs with early stopping. Finally, a bidirectional LSTM model will attempted

In [81]:
```
model = Sequential()
model.add(Embedding(input_dim = vocab_size, input_length = max_length, output_dim = 32
model.add(LSTM(32))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 25, 32)            571040

 lstm (LSTM)                 (None, 32)                8320

 dense (Dense)               (None, 32)                1056

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 580449 (2.21 MB)
Trainable params: 580449 (2.21 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [115…
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [116…
```
callbacks1 = [ModelCheckpoint(filepath='best_model.h5', monitor='val_accuracy', save_b
              EarlyStopping(monitor='val_accuracy', patience=3, verbose=1)]
history1 = model.fit(X_train, y_train, batch_size=32, epochs=20, verbose=1, validation
                    callbacks = callbacks1)
```

```
Epoch 1/20
151/151 [==============================] - 14s 77ms/step - loss: 0.0261 - accuracy:
0.9881 - val_loss: 1.5513 - val_accuracy: 0.7494
Epoch 2/20
151/151 [==============================] - 3s 21ms/step - loss: 0.0292 - accuracy: 0.
9879 - val_loss: 1.4455 - val_accuracy: 0.7410
Epoch 3/20
151/151 [==============================] - 2s 13ms/step - loss: 0.0210 - accuracy: 0.
9896 - val_loss: 1.4077 - val_accuracy: 0.7494
Epoch 4/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0209 - accuracy: 0.9
885 - val_loss: 1.5845 - val_accuracy: 0.7485
Epoch 4: early stopping
```

In [84]:
```
results1 = model.evaluate(X_test, y_test)
```

```
47/47 [==============================] - 0s 3ms/step - loss: 0.6834 - accuracy: 0.759
5
```

In [85]:
```
model2 = Sequential()
model2.add(Embedding(input_dim = vocab_size, input_length = max_length, output_dim = 3
model2.add(LSTM(64))
model2.add(Dense(64, activation='relu'))
model2.add(Dense(1, activation='sigmoid'))
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, 25, 32)            571040    
                                                                 
 lstm_1 (LSTM)               (None, 64)                24832     
                                                                 
 dense_2 (Dense)             (None, 64)                4160      
                                                                 
 dense_3 (Dense)             (None, 1)                 65        
                                                                 
=================================================================
Total params: 600097 (2.29 MB)
Trainable params: 600097 (2.29 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [86]:

```
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [87]:

```
callbacks2 = [ModelCheckpoint(filepath='best_model2.h5', monitor='val_accuracy', save_
              EarlyStopping(monitor='val_accuracy', patience=3, verbose=1)]
history2 = model2.fit(X_train, y_train, batch_size=32, epochs=20, verbose=1, validatic
```

```
Epoch 1/20
151/151 [==============================] - 15s 79ms/step - loss: 0.5990 - accuracy: 
0.6661 - val_loss: 0.4649 - val_accuracy: 0.7810
Epoch 2/20
151/151 [==============================] - 4s 24ms/step - loss: 0.3052 - accuracy: 0.
8888 - val_loss: 0.5086 - val_accuracy: 0.7719
Epoch 3/20
151/151 [==============================] - 3s 16ms/step - loss: 0.1559 - accuracy: 0.
9531 - val_loss: 0.6110 - val_accuracy: 0.7677
Epoch 4/20
151/151 [==============================] - 3s 18ms/step - loss: 0.0888 - accuracy: 0.
9744 - val_loss: 0.8535 - val_accuracy: 0.7577
Epoch 5/20
151/151 [==============================] - 2s 14ms/step - loss: 0.0729 - accuracy: 0.
9813 - val_loss: 0.7873 - val_accuracy: 0.7644
Epoch 6/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0601 - accuracy: 0.
9833 - val_loss: 0.9511 - val_accuracy: 0.7544
Epoch 7/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0529 - accuracy: 0.
9856 - val_loss: 0.7983 - val_accuracy: 0.7644
Epoch 8/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0477 - accuracy: 0.
9850 - val_loss: 0.8499 - val_accuracy: 0.7452
Epoch 9/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0357 - accuracy: 0.9
875 - val_loss: 0.8883 - val_accuracy: 0.7535
Epoch 10/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0395 - accuracy: 0.9
858 - val_loss: 0.8639 - val_accuracy: 0.7386
Epoch 11/20
151/151 [==============================] - 1s 6ms/step - loss: 0.0262 - accuracy: 0.9
852 - val_loss: 1.5213 - val_accuracy: 0.7502
Epoch 12/20
151/151 [==============================] - 1s 6ms/step - loss: 0.0293 - accuracy: 0.9
875 - val_loss: 0.9469 - val_accuracy: 0.7685
Epoch 13/20
151/151 [==============================] - 1s 6ms/step - loss: 0.0316 - accuracy: 0.9
881 - val_loss: 1.1872 - val_accuracy: 0.7344
Epoch 14/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0285 - accuracy: 0.9
869 - val_loss: 1.2647 - val_accuracy: 0.7619
Epoch 15/20
151/151 [==============================] - 1s 7ms/step - loss: 0.0264 - accuracy: 0.9
867 - val_loss: 0.9419 - val_accuracy: 0.7552
Epoch 16/20
151/151 [==============================] - 1s 6ms/step - loss: 0.0339 - accuracy: 0.9
856 - val_loss: 1.4521 - val_accuracy: 0.7494
Epoch 17/20
151/151 [==============================] - 1s 6ms/step - loss: 0.0244 - accuracy: 0.9
875 - val_loss: 1.5505 - val_accuracy: 0.7444
Epoch 18/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0195 - accuracy: 0.9
881 - val_loss: 1.7053 - val_accuracy: 0.7452
Epoch 19/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0187 - accuracy: 0.9
890 - val_loss: 1.7409 - val_accuracy: 0.7386
Epoch 20/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0237 - accuracy: 0.9
873 - val_loss: 1.4688 - val_accuracy: 0.7527
```

In [88]:

```
results2 = model2.evaluate(X_test, y_test)
```

```
47/47 [==============================] - 0s 3ms/step - loss: 1.5457 - accuracy: 0.738
8
```

In [89]:

```
model3 = Sequential()
model3.add(Embedding(input_dim = vocab_size, input_length = max_length, output_dim = 3
model3.add(LSTM(128))
model3.add(Dense(128, activation='relu'))
```

```python
model3.add(Dense(1, activation='sigmoid'))
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 25, 32)            571040    
                                                                 
 lstm_2 (LSTM)               (None, 128)               82432     
                                                                 
 dense_4 (Dense)             (None, 128)               16512     
                                                                 
 dense_5 (Dense)             (None, 1)                 129       
                                                                 
=================================================================
Total params: 670113 (2.56 MB)
Trainable params: 670113 (2.56 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [90]:

```python
model3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [91]:

```python
callbacks3 = [ModelCheckpoint(filepath='best_model3.h5', monitor='val_accuracy', save_
              EarlyStopping(monitor='val_accuracy', patience=3, verbose=1)]
history3 = model3.fit(X_train, y_train, batch_size=32, epochs=20, verbose=1, validatio
```

```
Epoch 1/20
151/151 [==============================] - 15s 81ms/step - loss: 0.5742 - accuracy:
0.6911 - val_loss: 0.4992 - val_accuracy: 0.7777
Epoch 2/20
151/151 [==============================] - 3s 19ms/step - loss: 0.2900 - accuracy: 0.
8929 - val_loss: 0.5082 - val_accuracy: 0.7843
Epoch 3/20
151/151 [==============================] - 2s 16ms/step - loss: 0.1410 - accuracy: 0.
9536 - val_loss: 0.6284 - val_accuracy: 0.7652
Epoch 4/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0912 - accuracy: 0.
9727 - val_loss: 0.7721 - val_accuracy: 0.7569
Epoch 5/20
151/151 [==============================] - 2s 13ms/step - loss: 0.0743 - accuracy: 0.
9792 - val_loss: 0.7358 - val_accuracy: 0.7510
Epoch 6/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0592 - accuracy: 0.
9835 - val_loss: 0.8825 - val_accuracy: 0.7494
Epoch 7/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0552 - accuracy: 0.9
842 - val_loss: 1.0059 - val_accuracy: 0.7552
Epoch 8/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0374 - accuracy: 0.9
873 - val_loss: 1.0487 - val_accuracy: 0.7602
Epoch 9/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0329 - accuracy: 0.9
869 - val_loss: 1.1738 - val_accuracy: 0.7669
Epoch 10/20
151/151 [==============================] - 1s 7ms/step - loss: 0.0267 - accuracy: 0.9
888 - val_loss: 1.9374 - val_accuracy: 0.7386
Epoch 11/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0319 - accuracy: 0.9
867 - val_loss: 1.7149 - val_accuracy: 0.7361
Epoch 12/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0537 - accuracy: 0.9
829 - val_loss: 1.2722 - val_accuracy: 0.7427
Epoch 13/20
151/151 [==============================] - 1s 7ms/step - loss: 0.0349 - accuracy: 0.9
863 - val_loss: 1.3184 - val_accuracy: 0.7627
Epoch 14/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0376 - accuracy: 0.
9863 - val_loss: 1.3030 - val_accuracy: 0.7677
Epoch 15/20
151/151 [==============================] - 1s 6ms/step - loss: 0.0256 - accuracy: 0.9
871 - val_loss: 1.5549 - val_accuracy: 0.7519
Epoch 16/20
151/151 [==============================] - 1s 6ms/step - loss: 0.0275 - accuracy: 0.9
890 - val_loss: 1.8549 - val_accuracy: 0.7119
Epoch 17/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0342 - accuracy: 0.9
867 - val_loss: 1.0300 - val_accuracy: 0.7452
Epoch 18/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0274 - accuracy: 0.9
871 - val_loss: 1.2767 - val_accuracy: 0.7485
Epoch 19/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0242 - accuracy: 0.9
879 - val_loss: 1.3288 - val_accuracy: 0.7527
Epoch 20/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0221 - accuracy: 0.9
885 - val_loss: 1.6520 - val_accuracy: 0.7502
```

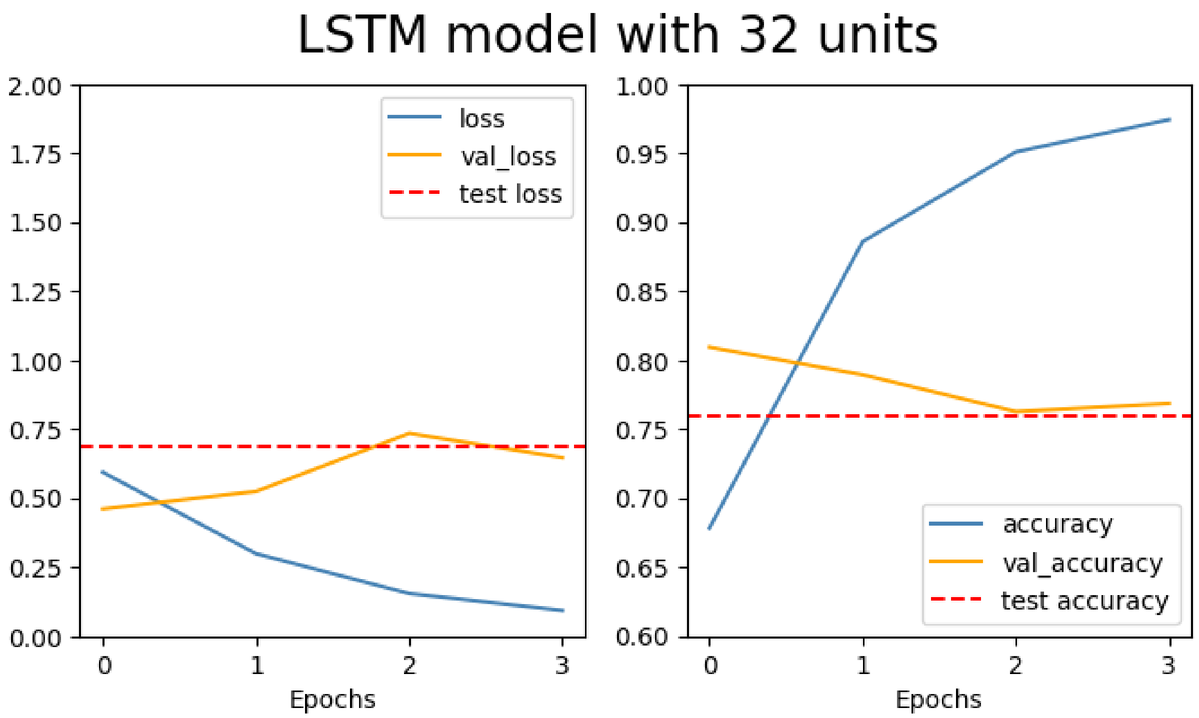
In [92]:
```
results3 = model3.evaluate(X_test, y_test)
```

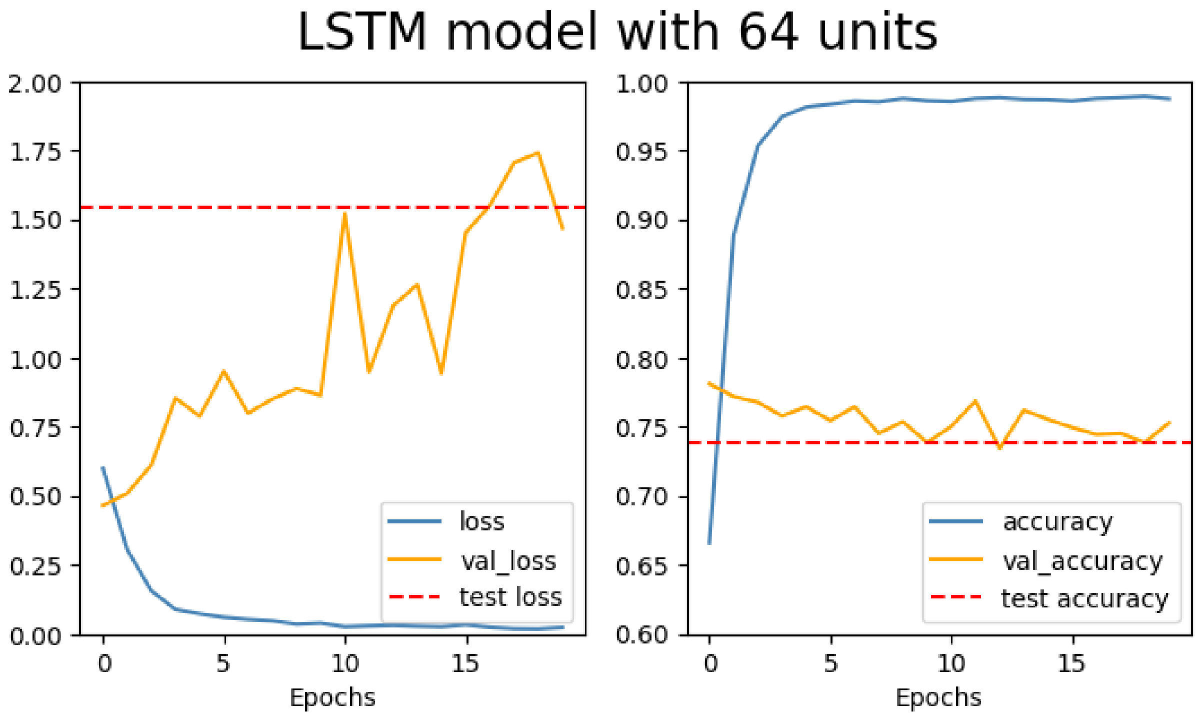
```
47/47 [==============================] - 0s 3ms/step - loss: 1.7631 - accuracy: 0.730
2
```

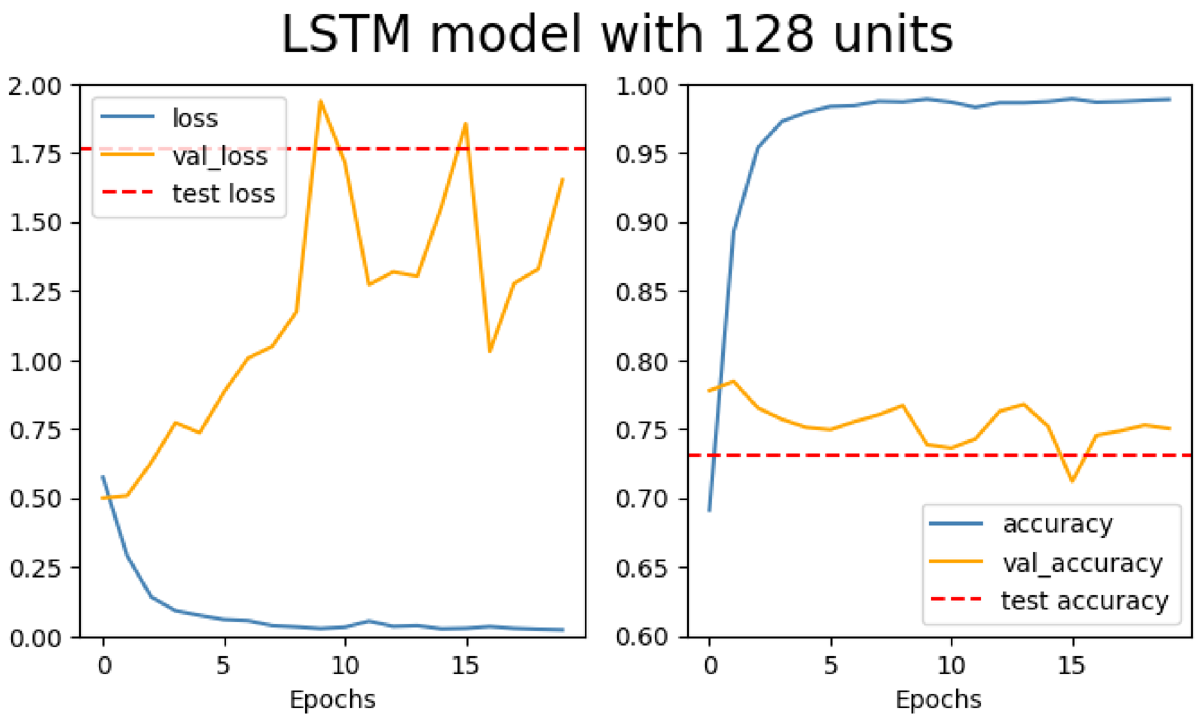
Visualizing the LSTM Models

In [93]:
```
def plot_loss_acc(training, results, model_name):
  fig, ax = plt.subplots(1, 2, figsize=(8,4), sharey=False)
  ax[0].plot(training.history['loss'], color='steelblue')
  ax[0].plot(training.history['val_loss'], color='orange')
  ax[0].axhline(y=results[0], color='red', linestyle='dashed')
  ax[0].set_ylim(0, 2)
  ax[0].legend(['loss', 'val_loss', 'test loss'])
  ax[0].set_xlabel('Epochs')

  ax[1].plot(training.history['accuracy'], color='steelblue')
  ax[1].plot(training.history['val_accuracy'], color='orange')
  ax[1].axhline(y=results[1], color='red', linestyle='dashed')
  ax[1].set_ylim(0.6, 1)
  ax[1].legend(['accuracy', 'val_accuracy','test accuracy'])
  ax[1].set_xlabel('Epochs')

  fig.suptitle(f"{model_name}", fontsize=20,
               verticalalignment='top')
  plt.show()
```

In [94]:
```
history = [history1, history2, history3]
results = [results1, results2, results3]
model_name = ['LSTM model with 32 units', 'LSTM model with 64 units', 'LSTM model with
for i in range(3):
    plot_loss_acc(history[i], results[i], model_name[i])
```







Bidirectional LSTM Models

In [95]:
```
model_bi = Sequential()
model_bi.add(Embedding(input_dim = vocab_size, input_length = max_length, output_dim =
model_bi.add(Bidirectional(LSTM(64)))
model_bi.add(Dense(32, activation='relu'))
model_bi.add(Dense(1, activation='sigmoid'))
model_bi.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, 25, 32)            571040    
                                                                 
 bidirectional (Bidirection  (None, 128)               49664     
 al)                                                             
                                                                 
 dense_6 (Dense)             (None, 32)                4128      
                                                                 
 dense_7 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 624865 (2.38 MB)
Trainable params: 624865 (2.38 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [96]:

```
model_bi.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [97]:

```
callbacks_bi = [ModelCheckpoint(filepath='best_model_bi.h5', monitor='val_accuracy', s
              EarlyStopping(monitor='val_accuracy', patience=3, verbose=1)]
history_bi = model_bi.fit(X_train, y_train, batch_size=32, epochs=20, verbose=1, valic
```

```
Epoch 1/20
151/151 [==============================] - 16s 80ms/step - loss: 0.5831 - accuracy:
0.6857 - val_loss: 0.4483 - val_accuracy: 0.8035
Epoch 2/20
151/151 [==============================] - 4s 26ms/step - loss: 0.2846 - accuracy: 0.
8854 - val_loss: 0.4778 - val_accuracy: 0.7835
Epoch 3/20
151/151 [==============================] - 2s 16ms/step - loss: 0.1304 - accuracy: 0.
9533 - val_loss: 0.5811 - val_accuracy: 0.7552
Epoch 4/20
151/151 [==============================] - 3s 19ms/step - loss: 0.0798 - accuracy: 0.
9758 - val_loss: 0.7220 - val_accuracy: 0.7444
Epoch 5/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0547 - accuracy: 0.
9829 - val_loss: 0.6927 - val_accuracy: 0.7594
Epoch 6/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0517 - accuracy: 0.
9833 - val_loss: 0.7584 - val_accuracy: 0.7336
Epoch 7/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0438 - accuracy: 0.9
840 - val_loss: 0.7918 - val_accuracy: 0.7527
Epoch 8/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0403 - accuracy: 0.
9829 - val_loss: 0.7823 - val_accuracy: 0.7744
Epoch 9/20
151/151 [==============================] - 1s 10ms/step - loss: 0.0373 - accuracy: 0.
9852 - val_loss: 0.9735 - val_accuracy: 0.7544
Epoch 10/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0336 - accuracy: 0.
9854 - val_loss: 0.9799 - val_accuracy: 0.7660
Epoch 11/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0322 - accuracy: 0.9
860 - val_loss: 0.9624 - val_accuracy: 0.7494
Epoch 12/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0295 - accuracy: 0.9
873 - val_loss: 0.9067 - val_accuracy: 0.7460
Epoch 13/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0273 - accuracy: 0.9
867 - val_loss: 0.9637 - val_accuracy: 0.7494
Epoch 14/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0254 - accuracy: 0.9
869 - val_loss: 0.9551 - val_accuracy: 0.7652
Epoch 15/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0247 - accuracy: 0.9
877 - val_loss: 0.8301 - val_accuracy: 0.7602
Epoch 16/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0282 - accuracy: 0.9
871 - val_loss: 1.1830 - val_accuracy: 0.7219
Epoch 17/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0231 - accuracy: 0.
9881 - val_loss: 1.1747 - val_accuracy: 0.7502
Epoch 18/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0208 - accuracy: 0.9
879 - val_loss: 1.1256 - val_accuracy: 0.7585
Epoch 19/20
151/151 [==============================] - 2s 12ms/step - loss: 0.0207 - accuracy: 0.
9898 - val_loss: 1.0720 - val_accuracy: 0.7544
Epoch 20/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0202 - accuracy: 0.
9883 - val_loss: 1.2313 - val_accuracy: 0.7519
```

In [98]:
```
results_bi = model_bi.evaluate(X_test, y_test)
```

```
47/47 [==============================] - 0s 4ms/step - loss: 1.3149 - accuracy: 0.732
8
```

In [99]:
```
model_bi2 = Sequential()
model_bi2.add(Embedding(input_dim = vocab_size, input_length = max_length, output_dim
model_bi2.add(Bidirectional(LSTM(64)))
model_bi2.add(Dense(64, activation='relu'))
```

```
model_bi2.add(Dense(1, activation='sigmoid'))
model_bi2.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_4 (Embedding)     (None, 25, 32)            571040    
                                                                 
 bidirectional_1 (Bidirecti  (None, 128)               49664     
 onal)                                                           
                                                                 
 dense_8 (Dense)             (None, 64)                8256      
                                                                 
 dense_9 (Dense)             (None, 1)                 65        
                                                                 
=================================================================
Total params: 629025 (2.40 MB)
Trainable params: 629025 (2.40 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [100…

```
model_bi2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [101…

```
callbacks_bi2 = [ModelCheckpoint(filepath='best_model_bi2.h5', monitor='val_accuracy',
                 EarlyStopping(monitor='val_accuracy', patience=3, verbose=1)]
history_bi2 = model_bi2.fit(X_train, y_train, batch_size=32, epochs=20, verbose=1, val
```

```
Epoch 1/20
151/151 [==============================] - 16s 81ms/step - loss: 0.5868 - accuracy:
0.6853 - val_loss: 0.4375 - val_accuracy: 0.8077
Epoch 2/20
151/151 [==============================] - 3s 22ms/step - loss: 0.2876 - accuracy: 0.
8900 - val_loss: 0.4865 - val_accuracy: 0.7735
Epoch 3/20
151/151 [==============================] - 3s 16ms/step - loss: 0.1349 - accuracy: 0.
9492 - val_loss: 0.6108 - val_accuracy: 0.7627
Epoch 4/20
151/151 [==============================] - 2s 14ms/step - loss: 0.0758 - accuracy: 0.
9742 - val_loss: 0.7071 - val_accuracy: 0.7435
Epoch 5/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0581 - accuracy: 0.
9810 - val_loss: 0.7417 - val_accuracy: 0.7427
Epoch 6/20
151/151 [==============================] - 2s 12ms/step - loss: 0.0482 - accuracy: 0.
9825 - val_loss: 0.8446 - val_accuracy: 0.7552
Epoch 7/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0431 - accuracy: 0.9
846 - val_loss: 0.8230 - val_accuracy: 0.7560
Epoch 8/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0398 - accuracy: 0.9
848 - val_loss: 0.8559 - val_accuracy: 0.7510
Epoch 9/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0357 - accuracy: 0.
9860 - val_loss: 0.8877 - val_accuracy: 0.7669
Epoch 10/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0321 - accuracy: 0.
9856 - val_loss: 0.8601 - val_accuracy: 0.7427
Epoch 11/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0311 - accuracy: 0.9
865 - val_loss: 0.8661 - val_accuracy: 0.7560
Epoch 12/20
151/151 [==============================] - 1s 10ms/step - loss: 0.0310 - accuracy: 0.
9856 - val_loss: 0.8316 - val_accuracy: 0.7477
Epoch 13/20
151/151 [==============================] - 2s 12ms/step - loss: 0.0275 - accuracy: 0.
9871 - val_loss: 1.1676 - val_accuracy: 0.7302
Epoch 14/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0263 - accuracy: 0.9
865 - val_loss: 0.9086 - val_accuracy: 0.7477
Epoch 15/20
151/151 [==============================] - 2s 9ms/step - loss: 0.0245 - accuracy: 0.9
883 - val_loss: 1.0579 - val_accuracy: 0.7502
Epoch 16/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0229 - accuracy: 0.
9877 - val_loss: 1.0289 - val_accuracy: 0.7510
Epoch 17/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0210 - accuracy: 0.
9890 - val_loss: 1.2512 - val_accuracy: 0.7377
Epoch 18/20
151/151 [==============================] - 1s 10ms/step - loss: 0.0196 - accuracy: 0.
9888 - val_loss: 1.4129 - val_accuracy: 0.7435
Epoch 19/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0207 - accuracy: 0.9
894 - val_loss: 1.2441 - val_accuracy: 0.7510
Epoch 20/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0221 - accuracy: 0.9
890 - val_loss: 1.2074 - val_accuracy: 0.7519
```

In [102…

```
results_bi2 = model_bi2.evaluate(X_test, y_test)
```

```
47/47 [==============================] - 0s 4ms/step - loss: 1.3518 - accuracy: 0.743
5
```

In [103…

```
model_bi3 = Sequential()
model_bi3.add(Embedding(input_dim = vocab_size, input_length = max_length, output_dim
model_bi3.add(Bidirectional(LSTM(64)))
model_bi3.add(Dense(128, activation='relu'))
```

```
model_bi3.add(Dense(1, activation='sigmoid'))
model_bi3.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 25, 32)            571040

 bidirectional_2 (Bidirecti  (None, 128)               49664
 onal)

 dense_10 (Dense)            (None, 128)               16512

 dense_11 (Dense)            (None, 1)                 129

=================================================================
Total params: 637345 (2.43 MB)
Trainable params: 637345 (2.43 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [104…

```
model_bi3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [105…

```
callbacks_bi3 = [ModelCheckpoint(filepath='best_model_bi3.h5', monitor='val_accuracy',
              EarlyStopping(monitor='val_accuracy', patience=3, verbose=1)]
history_bi3 = model_bi3.fit(X_train, y_train, batch_size=32, epochs=20, verbose=1, val
```

```
Epoch 1/20
151/151 [==============================] - 17s 85ms/step - loss: 0.5773 - accuracy: 0.6942 - val_loss: 0.5208 - val_accuracy: 0.7769
Epoch 2/20
151/151 [==============================] - 3s 17ms/step - loss: 0.2749 - accuracy: 0.8921 - val_loss: 0.4835 - val_accuracy: 0.7827
Epoch 3/20
151/151 [==============================] - 2s 14ms/step - loss: 0.1236 - accuracy: 0.9588 - val_loss: 0.5905 - val_accuracy: 0.7594
Epoch 4/20
151/151 [==============================] - 2s 12ms/step - loss: 0.0801 - accuracy: 0.9744 - val_loss: 0.6931 - val_accuracy: 0.7644
Epoch 5/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0585 - accuracy: 0.9813 - val_loss: 0.8146 - val_accuracy: 0.7386
Epoch 6/20
151/151 [==============================] - 2s 15ms/step - loss: 0.0532 - accuracy: 0.9827 - val_loss: 0.7196 - val_accuracy: 0.7352
Epoch 7/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0431 - accuracy: 0.9850 - val_loss: 0.7851 - val_accuracy: 0.7544
Epoch 8/20
151/151 [==============================] - 2s 12ms/step - loss: 0.0414 - accuracy: 0.9848 - val_loss: 0.7654 - val_accuracy: 0.7585
Epoch 9/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0359 - accuracy: 0.9863 - val_loss: 0.8339 - val_accuracy: 0.7352
Epoch 10/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0321 - accuracy: 0.9863 - val_loss: 0.8420 - val_accuracy: 0.7602
Epoch 11/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0311 - accuracy: 0.9854 - val_loss: 0.9292 - val_accuracy: 0.7369
Epoch 12/20
151/151 [==============================] - 2s 11ms/step - loss: 0.0294 - accuracy: 0.9863 - val_loss: 0.8961 - val_accuracy: 0.7435
Epoch 13/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0270 - accuracy: 0.9877 - val_loss: 0.9017 - val_accuracy: 0.7452
Epoch 14/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0247 - accuracy: 0.9877 - val_loss: 1.0039 - val_accuracy: 0.7477
Epoch 15/20
151/151 [==============================] - 1s 10ms/step - loss: 0.0226 - accuracy: 0.9881 - val_loss: 0.9609 - val_accuracy: 0.7669
Epoch 16/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0220 - accuracy: 0.9888 - val_loss: 1.3046 - val_accuracy: 0.7369
Epoch 17/20
151/151 [==============================] - 2s 10ms/step - loss: 0.0221 - accuracy: 0.9885 - val_loss: 1.0886 - val_accuracy: 0.7635
Epoch 18/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0211 - accuracy: 0.9885 - val_loss: 1.1050 - val_accuracy: 0.7469
Epoch 19/20
151/151 [==============================] - 1s 9ms/step - loss: 0.0200 - accuracy: 0.9898 - val_loss: 1.7311 - val_accuracy: 0.7344
Epoch 20/20
151/151 [==============================] - 1s 8ms/step - loss: 0.0300 - accuracy: 0.9873 - val_loss: 1.3921 - val_accuracy: 0.7477
```

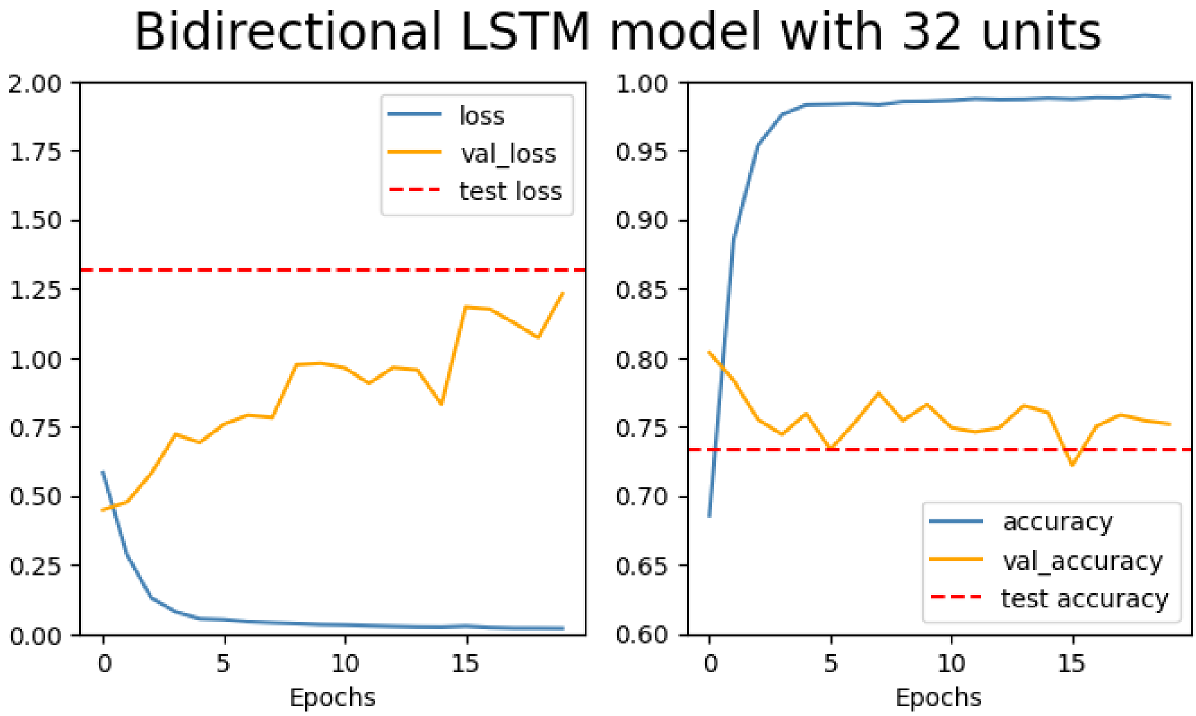
In [106…

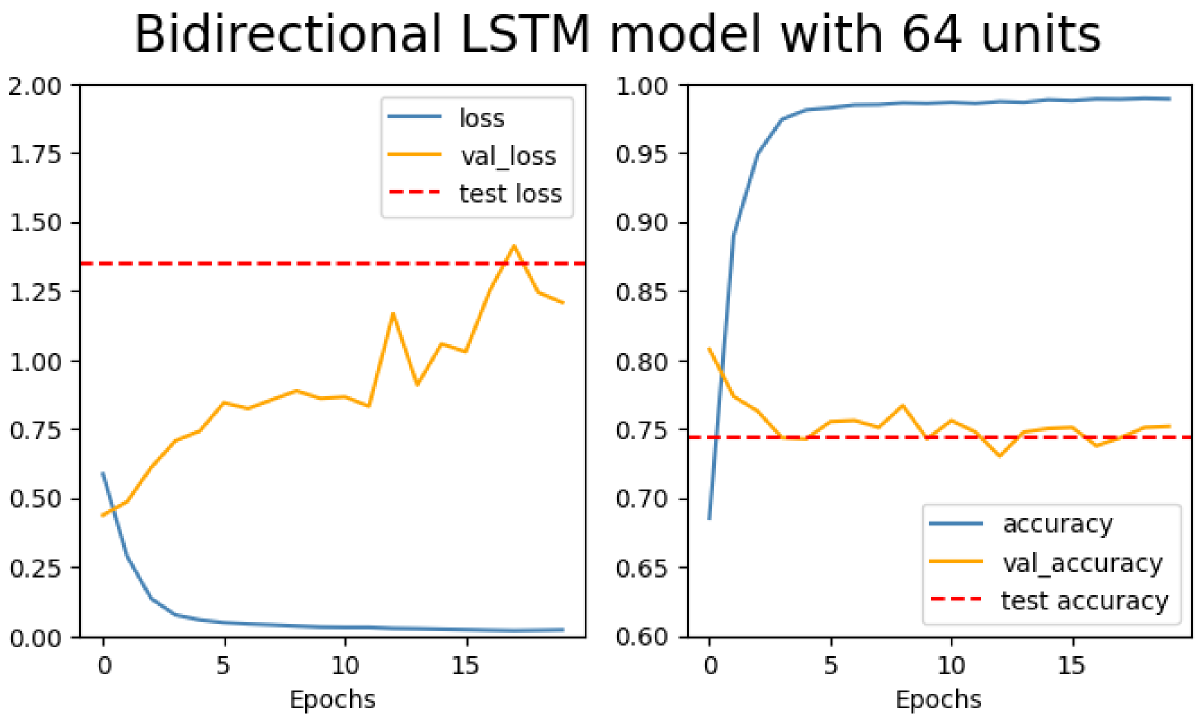
```
results_bi3 = model_bi3.evaluate(X_test, y_test)
```

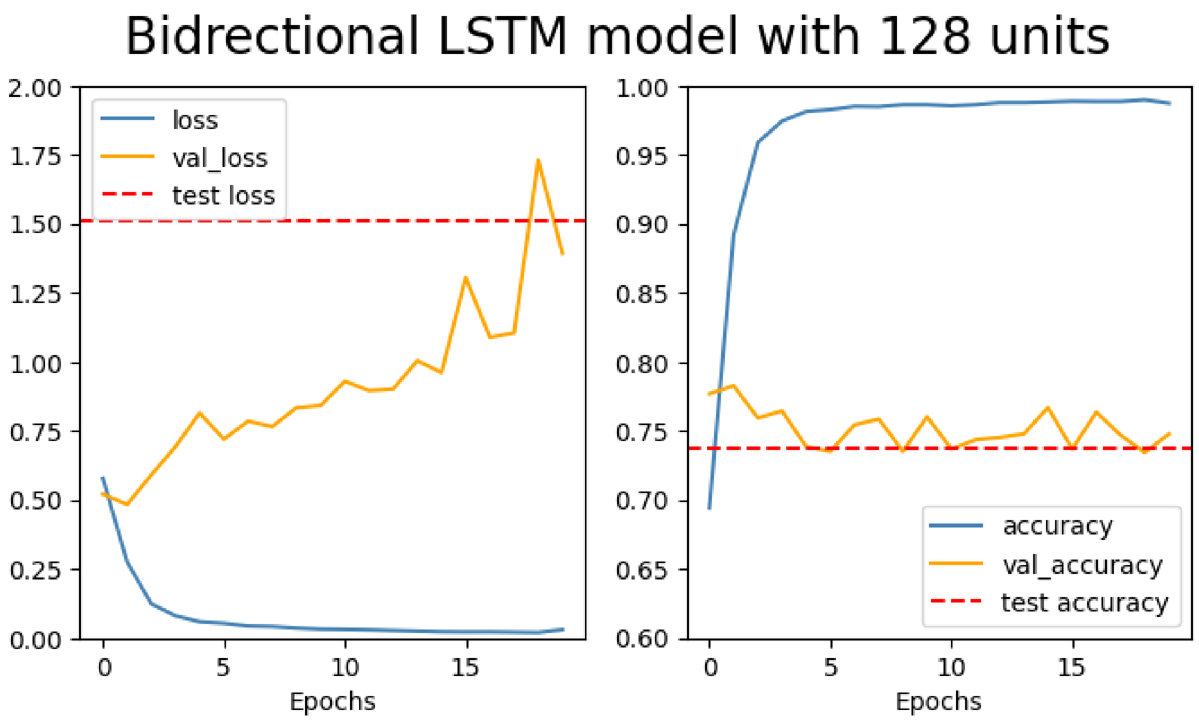
```
47/47 [==============================] - 0s 4ms/step - loss: 1.5109 - accuracy: 0.7375
```

In [107…

```
history = [history_bi, history_bi2, history_bi3]
results = [results_bi, results_bi2, results_bi3]
model_name = ['Bidirectional LSTM model with 32 units', 'Bidirectional LSTM model with
for i in range(3):
    plot_loss_acc(history[i], results[i], model_name[i])
```


Bidirectional LSTM model with 32 units
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
Epochs
loss
val_loss
test loss
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
5
10
15
Epochs
accuracy
val_accuracy
test accuracy


Bidirectional LSTM model with 64 units
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
Epochs
loss
val_loss
test loss
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
5
10
15
Epochs
accuracy
val_accuracy
test accuracy



In [109…
```
# Comparing loss and accuracy with test data

loss = np.round([results1[0], results2[0], results3[0], results_bi[0], results_bi2[0],
acc = np.round([results1[1], results2[1], results3[1], results_bi[1], results_bi2[1],

model_eval_df = pd.DataFrame({"loss": loss, "accuracy": acc},
                             index = ['LSTM 32 filters', 'LSTM 64 filters', 'LSTM 128
                                      'Bidirectional LSTM 32 filters', 'Bidirectional
                                      'Bidrectional LSTM model 128 filters'])
model_eval_df
```

Out[109]:

	loss	accuracy
LSTM 32 filters	0.683	0.759
LSTM 64 filters	1.546	0.739
LSTM 128 filters	1.763	0.730
Bidirectional LSTM 32 filters	1.315	0.733
Bidirectional LSTM model 64 filters	1.352	0.744
Bidrectional LSTM model 128 filters	1.511	0.738

Results

The best performing model is the Bidirectional LSTM model with 64 units.

Submission

In [117…
```
data_test = pd.read_csv('/kaggle/input/nlp-getting-started/test.csv')
data_test.head()
```

Out[117]:

	id	keyword	location	text
0	0	NaN	NaN	Just happened a terrible car crash
1	2	NaN	NaN	Heard about #earthquake is different cities, s...
2	3	NaN	NaN	there is a forest fire at spot pond, geese are...
3	9	NaN	NaN	Apocalypse lighting. #Spokane #wildfires
4	11	NaN	NaN	Typhoon Soudelor kills 28 in China and Taiwan

In [118…

```
data_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        3263 non-null   int64
 1   keyword   3237 non-null   object
 2   location  2158 non-null   object
 3   text      3263 non-null   object
dtypes: int64(1), object(3)
memory usage: 102.1+ KB
```

In [119…

```
id = data_test['id']
id.shape
```

Out[119]:

```
(3263,)
```

In [120…

```
data_test.nunique()
```

Out[120]:

```
id          3263
keyword      221
location    1602
text        3243
dtype: int64
```

In [121…

```
test_clean = data_test['text'].apply(tweet_cleaner)
test_clean
```

Out[121]:

```
0                              happened terrible car crash
1       heard earthquake different cities stay safe ev...
2       forest fire spot pond geese fleeing across str...
3                   apocalypse lighting spokane wildfires
4                  typhoon soudelor kills 28 china taiwan
                              ...
3258    earthquake safety los angeles safety fasteners...
3259    storm ri worse last hurricane cityamp3others h...
3260                          green line derailment chicago
3261               meg issues hazardous weather outlook hwo
3262    cityofcalgary activated municipal emergency pl...
Name: text, Length: 3263, dtype: object
```

In [122…

```
X_submission = tokenizer.texts_to_sequences(test_clean)
X_submission = pad_sequences(X_submission, padding='post', maxlen=max_length)
```

In [123…

```
X_submission.shape
```

Out[123]:

```
(3263, 25)
```

In [125…

```
target_1 = model.predict(X_submission)
target_1 = target_1.flatten()
target_1 = np.round(target_1, 0)
target_1.shape
```

```
102/102 [==============================] - 1s 2ms/step
```

Out[125]:

```
(3263,)
```

```
In [124…  target_2 = model_bi2.predict(X_submission)
          target_2 = target_2.flatten()
          target_2 = np.round(target_2, 0)
          target_2.shape
```

```
102/102 [==============================] - 1s 3ms/step
Out[124]: (3263,)
```

```
In [126…  id = data_test['id']
          id = np.array(id)
          id.shape
```

```
Out[126]: (3263,)
```

```
In [132…  data_test['predict_1'] = target_1
          data_test['predict_2'] = target_2
          data_test[['text', 'predict_1', 'predict_2']].head(10)
```

Out[132]:

	text	predict_1	predict_2
0	Just happened a terrible car crash	0.0	0.0
1	Heard about #earthquake is different cities, s...	0.0	0.0
2	there is a forest fire at spot pond, geese are...	1.0	1.0
3	Apocalypse lighting. #Spokane #wildfires	1.0	1.0
4	Typhoon Soudelor kills 28 in China and Taiwan	1.0	1.0
5	We're shaking...It's an earthquake	0.0	1.0
6	They'd probably still show more life than Arse...	0.0	0.0
7	Hey! How are you?	0.0	0.0
8	What a nice hat?	0.0	0.0
9	Fuck off!	0.0	0.0

Looking at a sample of the predictions, it is clear the model is not perfect. The second tweet should have been classified as '1'. The first tweet illustrates one of the challenges of this classification task: it is predicted to not be a disaster, and while it is not a natural disaster, it is questionable if 'terrible car crash' could could as disaster. The two models differ in their predictions for 'We're shaking...It's an earthquake', with the LSTM model labeling it non-disaster, and the bidirectional LSTM model labeling it a disaster. The other samples appear to be classified correctly.

```
In [134…  np.savetxt(
              'submission2.csv',
              np.rec.fromarrays([id, target_2]),
              fmt=['%s', '%g'],
              delimiter=',',
              header='id,target',
              comments='',
          )

          # Look at the first few predictions
          !head submission.csv
```

```
head: cannot open 'submission.csv' for reading: No such file or directory
```

Results

Both models had similar performance in the Kaggle Competition: the LSTM model had 0.7634 and the bidirectional LSTM model 0.76708. In both cases the results were slightly better than the model evaluation. In the initial evaluation the LSTM model had lower loss and higher accuracy, but it only trained over 4 epochs due to early stopping, so it is possible the fewer training epochs can account for the small difference in results.

Conclusions

Both the LSTM and Bidirectional LSTM were able to correctly classify the tweets nearly three-quarters of the time. There was no a significant variation in performance as the number of LSTM filters increased from 32 to 64 to 128. In the future it would be good to try more complex NLP models, as well as different approaches to processing the textual data.

Sources

Word2Vec

- https://www.analyticsvidhya.com/blog/2023/07/step-by-step-guide-to-word2vec-with-gensim/
- https://www.analyticsvidhya.com/blog/2021/07/word2vec-for-word-embeddings-a-beginners-guide/
- https://www.kaggle.com/code/lystdo/lstm-with-word2vec-embeddings
- https://www.kaggle.com/code/guichristmann/lstm-classification-model-with-word2vec