week4

April 12, 2024

0.1 Week 4 : NLP Disaster Tweets Kaggle Mini-Project

Student, University of Colorado Boulder

0.2 Step 1: Brief description of the problem and data

We are provided with tweets which are classified as 1 if they are about real disasters and 0 if they are not. We will use these as training data. Intent of the project is to build a machine learning model that predicts which tweets are about real disasters and which one's aren't. We will develop this model and test them on the test data.

The details of the project is available in Kaggle competition https://www.kaggle.com/c/nlp-getting-started/overview

[1]:
```python
# Imports

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import nltk
from nltk.stem import WordNetLemmatizer

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

from copy import deepcopy

from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import AUC
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model
```

```
/Users/kkuruvad/Desktop/masters/UniversityOfColoradoBoulder/DTSA 5511
Introduction to Deep Learning/week4/venv/lib/python3.9/site-
packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports
OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'.
See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

Data Extraction User has to register on kaggle to get access to the dataset. This data is expected to be stored in data/nlp-getting-started folder

[2]:
```
data_path = 'data/nlp-getting-started/'
df_train = pd.read_csv(data_path+'train.csv')
df_test = pd.read_csv(data_path+'test.csv')

df_train.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
```

There are total of 7613 rows with 5 columns. 2 columns are of integer type and 3 are of type object (string). The id field is unique with values going from 0 to 7612.

[3]:
```
df_train.head(10)
```

[3]:
```
   id keyword location                                               text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M…
1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
2   5     NaN      NaN  All residents asked to 'shelter in place' are …
3   6     NaN      NaN  13,000 people receive #wildfires evacuation or…
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as …
5   8     NaN      NaN  #RockyFire Update => California Hwy. 20 closed…
6  10     NaN      NaN  #flood #disaster Heavy rain causes flash flood…
7  13     NaN      NaN  I'm on top of the hill and I can see a fire in…
8  14     NaN      NaN  There's an emergency evacuation happening now …
9  15     NaN      NaN  I'm afraid that the tornado is coming to our a…

   target
0       1
1       1
2       1
3       1
4       1
5       1
6       1
7       1
```

```
8        1
9        1
```

[4]:
```python
df_train.describe()
```

[4]:
```
                 id      target
count   7613.000000  7613.00000
mean    5441.934848     0.42966
std     3137.116090     0.49506
min        1.000000     0.00000
25%     2734.000000     0.00000
50%     5408.000000     0.00000
75%     8146.000000     1.00000
max    10873.000000     1.00000
```

[5]:
```python
df_test.info
```

[5]:
```
<bound method DataFrame.info of           id keyword location  \
0         0     NaN      NaN
1         2     NaN      NaN
2         3     NaN      NaN
3         9     NaN      NaN
4        11     NaN      NaN
…       …      …       …
3258  10861     NaN      NaN
3259  10865     NaN      NaN
3260  10868     NaN      NaN
3261  10874     NaN      NaN
3262  10875     NaN      NaN

                                                   text
0                    Just happened a terrible car crash
1     Heard about #earthquake is different cities, s…
2     there is a forest fire at spot pond, geese are…
3              Apocalypse lighting. #Spokane #wildfires
4         Typhoon Soudelor kills 28 in China and Taiwan
…                                                   …
3258  EARTHQUAKE SAFETY LOS ANGELES  ÛÒ SAFETY FASTE…
3259  Storm in RI worse than last hurricane. My city…
3260  Green Line derailment in Chicago http://t.co/U…
3261  MEG issues Hazardous Weather Outlook (HWO) htt…
3262  #CityofCalgary has activated its Municipal Eme…

[3263 rows x 4 columns]>
```

[6]:
```python
df_test.describe()
```

[6]:

```
                 id
count   3263.000000
mean    5427.152927
std     3146.427221
min        0.000000
25%     2683.000000
50%     5500.000000
75%     8176.000000
max    10875.000000
```

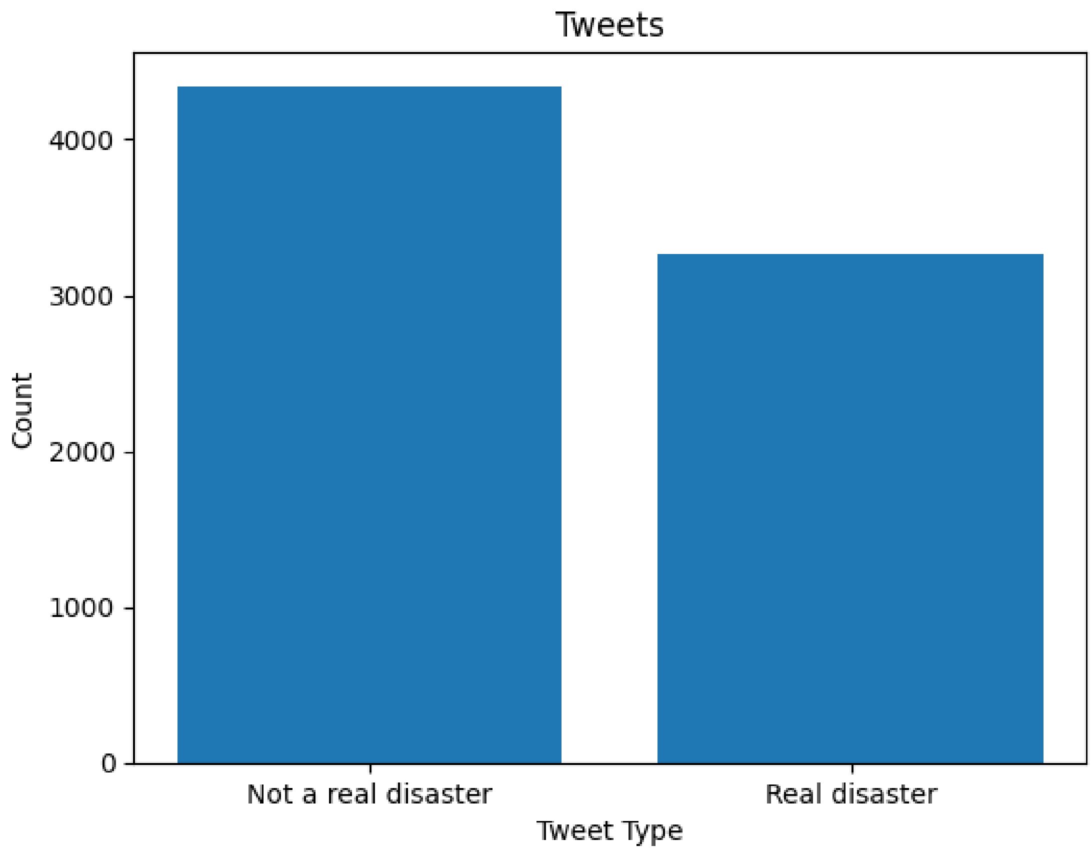
We have 3263 rows of data in the test dataframe and the structure is similar to the training data without the target column

0.3 Step 2: Exploratory Data Analysis (EDA) — Inspect, Visualize and Clean the Data

Let us look at distribution of disaster vs not disaster tweets in the training data

[7]:

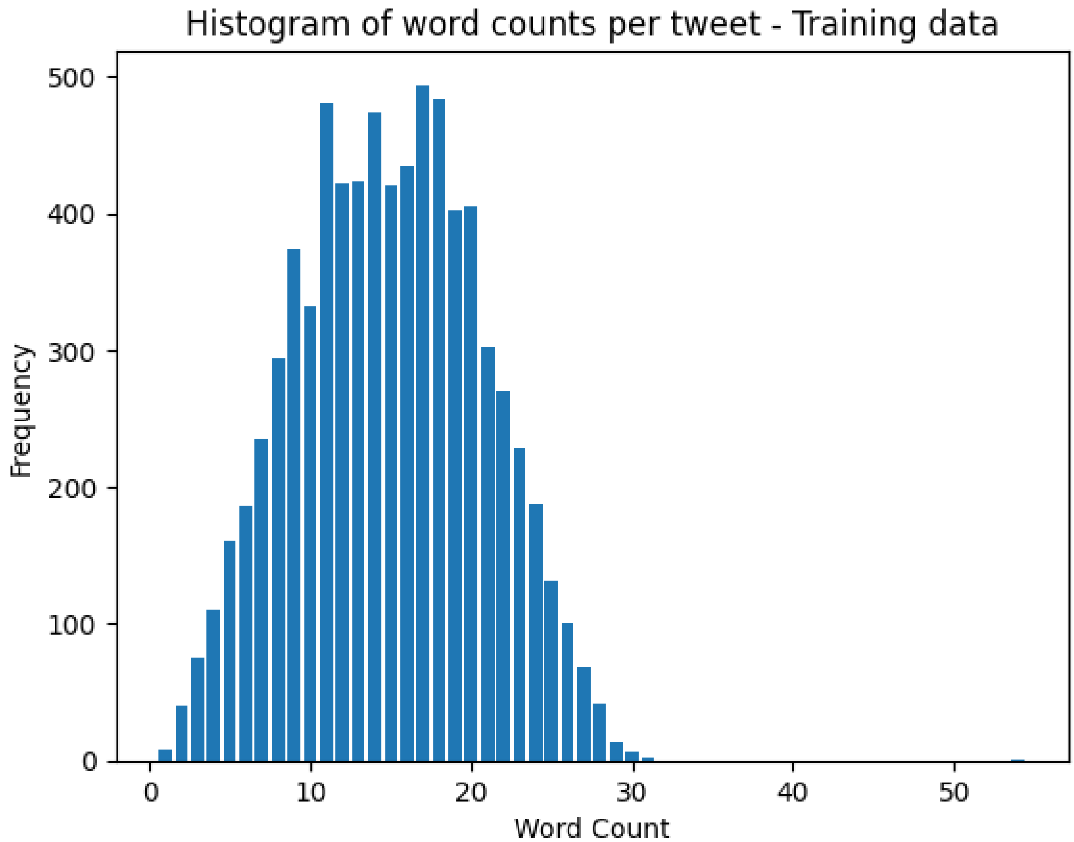
```
df_train['target'].plot.hist()

x = ["Not a real disaster", "Real disaster"]
y = [df_train['target'].value_counts()[0.0], df_train['target'].
 ↪value_counts()[1.0]]

plt.bar(x, y)
plt.title("Tweets")

plt.xlabel("Tweet Type")
plt.ylabel("Count")

plt.show()
```

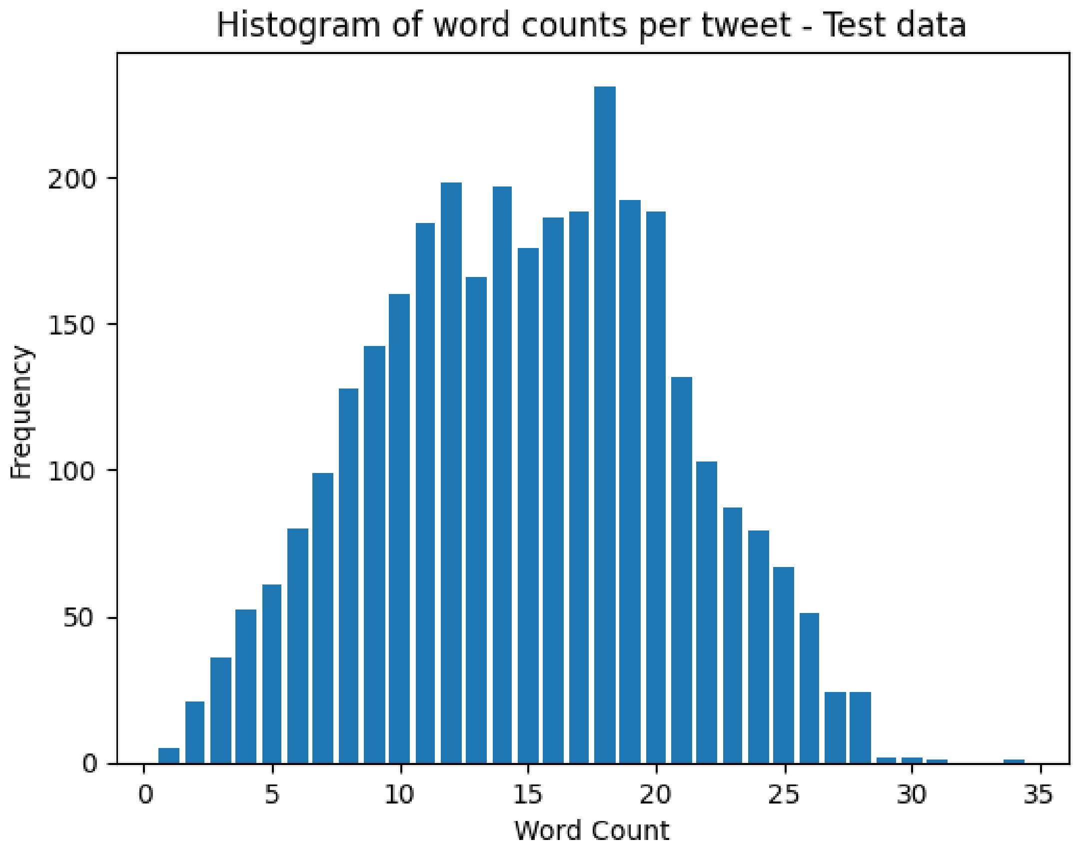


Let us look at the size of tweets

[8]:
```
def histogram_of_word_counts(strings, df_type):
    counts = {}
    for string in strings:
        words = string.split(' ')
        length = len(words)
        if length not in counts:
            counts[length] = 0
        counts[length] += 1
        #if length > 50:
        #    print(f'{string}: {words}')

    word_counts = list(counts.keys())

    counts = list(counts.values())

    # Create the histogram.
    plt.bar(word_counts, counts)
    plt.xlabel("Word Count")
```

```
    plt.ylabel("Frequency")
    plt.title(f"Histogram of word counts per tweet - {df_type} data")
    plt.show()

histogram_of_word_counts(df_train['text'], 'Training')
histogram_of_word_counts(df_test['text'], 'Test')
```





Let us look at some frequent words in the tweets

[9]:
```
stopwords = set(STOPWORDS)
stopwords.update(['u', 't', 'https', 'co', 'Û_', 'Û'])
text = " ".join(review for review in df_train.text)
# Generate a word cloud image
wordcloud = WordCloud(stopwords=stopwords, background_color="white").
 ↪generate(text)

# Display the generated image:
# the matplotlib way:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Cleaning We dont need the id, keyword and location fields so we can drop those columns in training dataframe. We need the id in test dataframe so we wont drop that.

[10]:
```
df_train = df_train.drop(columns=['id', 'location', 'keyword'])
df_test = df_test.drop(columns=['location', 'keyword'])
df_train.info
```

[10]:
```
<bound method DataFrame.info of
text  target
0     Our Deeds are the Reason of this #earthquake M…       1
1                Forest fire near La Ronge Sask. Canada       1
2     All residents asked to 'shelter in place' are …       1
3     13,000 people receive #wildfires evacuation or…       1
4     Just got sent this photo from Ruby #Alaska as …       1
…                                                   …     …
7608  Two giant cranes holding a bridge collapse int…       1
7609  @aria_ahrary @TheTawniest The out of control w…       1
7610  M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt…       1
7611  Police investigating after an e-bike collided …       1
7612  The Latest: More Homes Razed by Northern Calif…       1

[7613 rows x 2 columns]>
```

0.4 Step 3: Model Architecture

Before we start building tensorflow models, let us convert the data and pad if necessary. We will use GloVe to process texts to matrix form (word embedding)

3a. Download the necessary GloVe zip file

[26]:
```
!wget http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
```

```
--2024-04-10 10:18:21--  http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)… 171.64.64.22
Connecting to downloads.cs.stanford.edu
(downloads.cs.stanford.edu)|171.64.64.22|:80… connected.
HTTP request sent, awaiting response… 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip.1’

glove.6B.zip.1      100%[===================>] 822.24M   834KB/s    in 14m 26s

2024-04-10 10:32:46 (972 KB/s) - ‘glove.6B.zip.1’ saved [862182613/862182613]
```

3b. Extract the zip file

[18]:
```
!unzip glove.6B.zip
```

```
Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

3c. Form the GloVe word mappings. Let us use the 100 dimension file.

[11]:
```
def glove_word_mappings(filename):
    word_mapping = dict()
    with open(filename, 'r') as f:
        for line in f.readlines():
            try:
                words = line.split(' ')
                word_mapping[words[0]] = np.array(words[1:], dtype=float)
            except Exception as ex:
                print(f'failed to use {words[0]}: {ex}')
                break
    return word_mapping


glove_words = glove_word_mappings('glove.6B.100d.txt')
print(f'count of glove words {len(glove_words)}')
```

```
count of glove words 400000
```

3d. Download wordnet using nltk. We will be using this for the tokenizer

[12]:
```
nltk.download('wordnet')

regexp_tokenizer = nltk.RegexpTokenizer(r'\w+')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/kkuruvad/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

3e. Function to lemmatize the tweet/a given text. We will only use the lemmatized token if it is present in GloVe

[13]:
```
lemmatizer = WordNetLemmatizer()

def tweet_to_tokens(tweet):
    tokens = regexp_tokenizer.tokenize(tweet)
    tokens = [t.lower() for t in tokens]
    lemmatized_tokens = [lemmatizer.lemmatize(t) for t in tokens]
    return [t for t in lemmatized_tokens if t in glove_words]
```

3f. Function to convert the tokens into vectors

[14]:
```
def tokens_to_vectors(tokens):
    vectors = list()
    for token in tokens:
        if token not in glove_words:
            continue
        vectors.append(glove_words[token])
    return np.array(vectors, dtype=float)
```

3g. Split input data into training (70%) and validation (30%)

[15]:
```
m_df_train = df_train.sample(frac=1, random_state=142)
m_df_train.reset_index(drop=True, inplace=True)

train_validate_split = int(len(m_df_train)*0.7)

train, validate = m_df_train[:train_validate_split],␣
 ↪m_df_train[train_validate_split:]
len(train), len(validate)
```

[15]: (5329, 2284)

3h. Convert the training, validation and test dataframes into tokens

[16]:
```
def dataframe_to_X_y(df):
    y = None
    if 'target' in df:
        y = df['target'].to_numpy().astype(int)

    vectors = list()
    for tweet in df['text']:
        tokens = tweet_to_tokens(tweet)
        v = tokens_to_vectors(tokens)
        if v.shape[0] == 0:
```

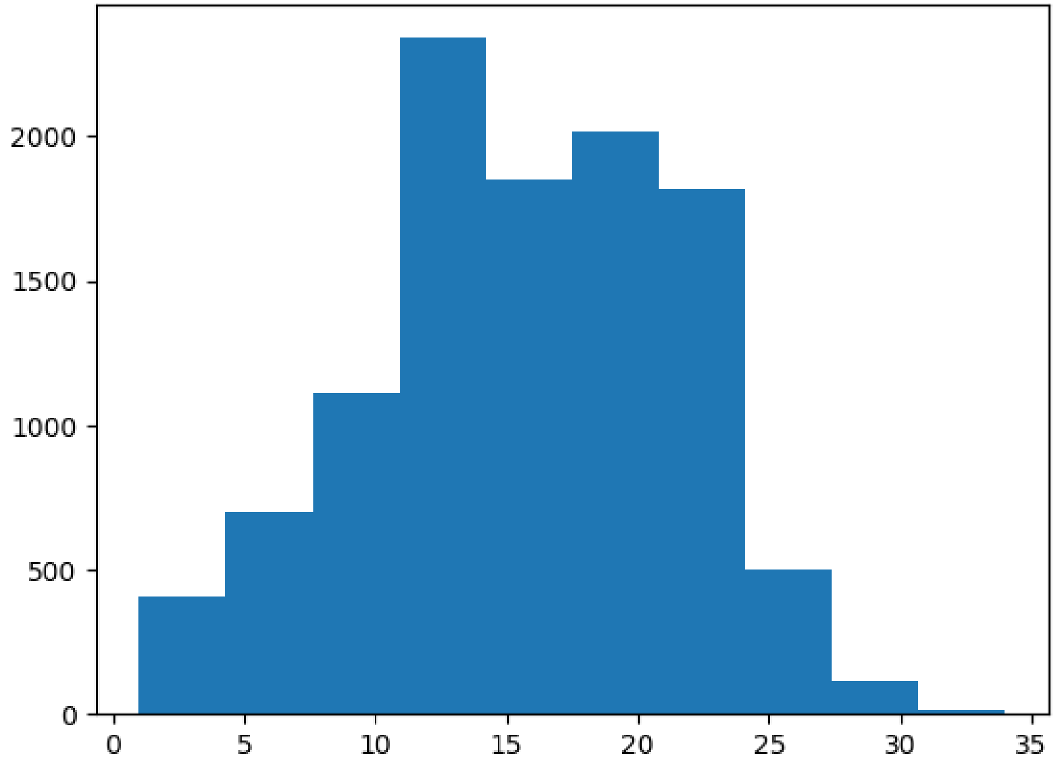
```
            vectors.append(np.zeros(shape=(1,100)))
        else:
            vectors.append(v)
    return vectors, y

df_test.info()
X_train, y_train = dataframe_to_X_y(train)
X_val, y_val = dataframe_to_X_y(validate)
X_test,_ = dataframe_to_X_y(df_test)

print(f'len(X_train): {len(X_train)}, len(y_train): {len(y_train)}, len(X_val):␣
 ↪{len(X_val)}, len(y_val): {len(y_val)}, len(X_test):{len(X_test)}')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      3263 non-null   int64
 1   text    3263 non-null   object
dtypes: int64(1), object(1)
memory usage: 51.1+ KB
len(X_train): 5329, len(y_train): 5329, len(X_val): 2284, len(y_val): 2284,
len(X_test):3263
```

3i. Check for the maximum vectors size of the tweet in all the 3 dataframes (training, validation and test) . We will use this later in padding

[17]:
```
sequence_lengths = list()
for i in range(len(X_train)):
    sequence_lengths.append(len(X_train[i]))
for i in range(len(X_val)):
    sequence_lengths.append(len(X_val[i]))
for i in range(len(X_test)):
    sequence_lengths.append(len(X_test[i]))

plt.hist(sequence_lengths)
pd.Series(sequence_lengths).describe()
```

[17]:
```
count    10876.000000
mean        15.636539
std          5.927161
min          1.000000
25%         12.000000
50%         16.000000
75%         20.000000
max         34.000000
dtype: float64
```



3j. Use the maximum size of vectors to determine padding. We will round the value and consider 35

[18]:
```python
def pad_X(X, length, dimension):
    X_copy = deepcopy(X)
    for i,x in enumerate(X):
        l = x.shape[0]
        padding = np.zeros(shape=(length-x.shape[0], dimension))
        X_copy[i] = np.concatenate([x, padding])
    return np.array(X_copy).astype(float)

#X_train.shape
X_train_pad = pad_X(X_train, 35, 100)
print(f'training: {X_train_pad.shape}')

X_val_pad = pad_X(X_val, 35, 100)
print(f'validation: {X_val_pad.shape}')

X_test_pad = pad_X(X_test, 35, 100)
print(f'test: {X_test_pad.shape}')
```

```
training: (5329, 35, 100)
validation: (2284, 35, 100)
```

```
test: (3263, 35, 100)
```

Models Now that our data is converted, we can start with tensorflow models

After padding our X has shape of 35, 100. So lets declare input layer of this shape LSTM of 64 units is good enough in most cases. So lets add that. Let us also add a drop out layer with 0.2.

Let us repeat the same layers again and finally flatten and apply sigmoid activation since we want the output to be 0 or 1

[19]:
```
model = Sequential([])

model.add(layers.Input(shape=(35,100)))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))

cp = ModelCheckpoint('tf_model/model_seq_1.keras', save_best_only=True)

model.compile(optimizer=Adam(learning_rate=0.0001), loss=BinaryCrossentropy(),
 ↪metrics=['Accuracy', AUC(name='auc')])

model.fit(X_train_pad, y_train, validation_data=(X_val_pad, y_val), epochs=20,
 ↪callbacks=[cp])
```

```
Epoch 1/20
167/167              5s 22ms/step -
Accuracy: 0.5982 - auc: 0.6246 - loss: 0.6557 - val_Accuracy: 0.7609 - val_auc:
0.8161 - val_loss: 0.5218
Epoch 2/20
167/167              4s 21ms/step -
Accuracy: 0.7674 - auc: 0.8276 - loss: 0.5050 - val_Accuracy: 0.7763 - val_auc:
0.8482 - val_loss: 0.4837
Epoch 3/20
167/167              4s 22ms/step -
Accuracy: 0.7987 - auc: 0.8520 - loss: 0.4593 - val_Accuracy: 0.7973 - val_auc:
0.8635 - val_loss: 0.4514
Epoch 4/20
167/167              4s 22ms/step -
Accuracy: 0.8052 - auc: 0.8683 - loss: 0.4383 - val_Accuracy: 0.8074 - val_auc:
0.8687 - val_loss: 0.4409
Epoch 5/20
167/167              4s 21ms/step -
Accuracy: 0.8015 - auc: 0.8634 - loss: 0.4428 - val_Accuracy: 0.8008 - val_auc:
0.8709 - val_loss: 0.4419
Epoch 6/20
```

```
167/167              4s 21ms/step -
Accuracy: 0.8143 - auc: 0.8781 - loss: 0.4189 - val_Accuracy: 0.8043 - val_auc:
0.8721 - val_loss: 0.4352
Epoch 7/20
167/167              4s 21ms/step -
Accuracy: 0.8098 - auc: 0.8758 - loss: 0.4245 - val_Accuracy: 0.8065 - val_auc:
0.8736 - val_loss: 0.4342
Epoch 8/20
167/167              4s 21ms/step -
Accuracy: 0.8197 - auc: 0.8751 - loss: 0.4191 - val_Accuracy: 0.7999 - val_auc:
0.8743 - val_loss: 0.4394
Epoch 9/20
167/167              4s 22ms/step -
Accuracy: 0.8138 - auc: 0.8847 - loss: 0.4145 - val_Accuracy: 0.8060 - val_auc:
0.8738 - val_loss: 0.4355
Epoch 10/20
167/167              4s 21ms/step -
Accuracy: 0.8259 - auc: 0.8876 - loss: 0.4019 - val_Accuracy: 0.8074 - val_auc:
0.8743 - val_loss: 0.4295
Epoch 11/20
167/167              3s 21ms/step -
Accuracy: 0.8297 - auc: 0.8910 - loss: 0.3983 - val_Accuracy: 0.8043 - val_auc:
0.8742 - val_loss: 0.4348
Epoch 12/20
167/167              4s 21ms/step -
Accuracy: 0.8304 - auc: 0.8917 - loss: 0.3956 - val_Accuracy: 0.8082 - val_auc:
0.8750 - val_loss: 0.4279
Epoch 13/20
167/167              4s 22ms/step -
Accuracy: 0.8221 - auc: 0.8896 - loss: 0.3990 - val_Accuracy: 0.8113 - val_auc:
0.8764 - val_loss: 0.4292
Epoch 14/20
167/167              4s 21ms/step -
Accuracy: 0.8244 - auc: 0.8883 - loss: 0.4015 - val_Accuracy: 0.8087 - val_auc:
0.8755 - val_loss: 0.4328
Epoch 15/20
167/167              4s 22ms/step -
Accuracy: 0.8355 - auc: 0.8957 - loss: 0.3816 - val_Accuracy: 0.8091 - val_auc:
0.8756 - val_loss: 0.4386
Epoch 16/20
167/167              4s 22ms/step -
Accuracy: 0.8319 - auc: 0.9023 - loss: 0.3753 - val_Accuracy: 0.8047 - val_auc:
0.8752 - val_loss: 0.4362
Epoch 17/20
167/167              4s 22ms/step -
Accuracy: 0.8456 - auc: 0.9034 - loss: 0.3714 - val_Accuracy: 0.8039 - val_auc:
0.8741 - val_loss: 0.4374
Epoch 18/20
```

```
167/167              4s 21ms/step -
Accuracy: 0.8465 - auc: 0.9062 - loss: 0.3674 - val_Accuracy: 0.8087 - val_auc:
0.8757 - val_loss: 0.4299
Epoch 19/20
167/167              4s 22ms/step -
Accuracy: 0.8403 - auc: 0.9040 - loss: 0.3722 - val_Accuracy: 0.8109 - val_auc:
0.8764 - val_loss: 0.4331
Epoch 20/20
167/167              4s 21ms/step -
Accuracy: 0.8406 - auc: 0.9005 - loss: 0.3760 - val_Accuracy: 0.8100 - val_auc:
0.8748 - val_loss: 0.4398
```

[19]:
```
<keras.src.callbacks.history.History at 0x1781fe160>
```

Let us load the best model and run predictions on the test set

[100]:
```python
best_model = load_model('tf_model/model_seq_1.keras')

predictions = (best_model.predict(X_test_pad) > 0.5).astype(int)
predictions
output_df = pd.DataFrame(df_test['id'])
output_df['target'] = predictions
out = output_df.to_csv('Disaster_tweets_RNN_seq_1.csv', index=False)
```

```
102/102              1s 6ms/step
```

The model got a score of 0.79773 which is not bad.

Parameter tuning Let us see if we can get a better score. We already know that the tweet 'target' - ie if a tweet is of a real disaster or not, is skewed with more zeros (not disaster) that ones (disaster). Let us use the weights parameter of the fit to see if we can get better results

[22]:
```python
disaster_counts = pd.Series(df_train['target']).value_counts()
disaster_counts
weights = {0:disaster_counts.sum()/disaster_counts[0], 1:disaster_counts.sum()/
 ↪disaster_counts[1]}
weights
```

[22]:
```
{0: 1.753339474896361, 1: 2.3274228064811986}
```

[102]:
```python
model = Sequential([])

model.add(layers.Input(shape=(35,100)))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
```

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

cp = ModelCheckpoint('tf_model/model.keras', save_best_only=True)

model.compile(optimizer=Adam(learning_rate=0.0001), loss=BinaryCrossentropy(),␣
↪metrics=['Accuracy', AUC(name='auc')])

model.fit(X_train_pad, y_train, validation_data=(X_val_pad, y_val), epochs=20,␣
↪callbacks=[cp], class_weight=weights)
```

```
Epoch 1/20
167/167            5s 22ms/step -
Accuracy: 0.6667 - auc: 0.7015 - loss: 1.3042 - val_Accuracy: 0.7487 - val_auc:
0.8175 - val_loss: 0.5237
Epoch 2/20
167/167            4s 22ms/step -
Accuracy: 0.7798 - auc: 0.8384 - loss: 0.9874 - val_Accuracy: 0.7785 - val_auc:
0.8495 - val_loss: 0.4775
Epoch 3/20
167/167            4s 22ms/step -
Accuracy: 0.7965 - auc: 0.8611 - loss: 0.9091 - val_Accuracy: 0.7885 - val_auc:
0.8631 - val_loss: 0.4644
Epoch 4/20
167/167            4s 22ms/step -
Accuracy: 0.7958 - auc: 0.8635 - loss: 0.9012 - val_Accuracy: 0.7973 - val_auc:
0.8647 - val_loss: 0.4458
Epoch 5/20
167/167            4s 22ms/step -
Accuracy: 0.8015 - auc: 0.8702 - loss: 0.8887 - val_Accuracy: 0.7999 - val_auc:
0.8683 - val_loss: 0.4511
Epoch 6/20
167/167            4s 23ms/step -
Accuracy: 0.8151 - auc: 0.8851 - loss: 0.8364 - val_Accuracy: 0.7964 - val_auc:
0.8695 - val_loss: 0.4529
Epoch 7/20
167/167            4s 23ms/step -
Accuracy: 0.8107 - auc: 0.8763 - loss: 0.8670 - val_Accuracy: 0.8060 - val_auc:
0.8709 - val_loss: 0.4363
Epoch 8/20
167/167            4s 23ms/step -
Accuracy: 0.8331 - auc: 0.8914 - loss: 0.8016 - val_Accuracy: 0.8087 - val_auc:
0.8718 - val_loss: 0.4384
Epoch 9/20
167/167            4s 22ms/step -
Accuracy: 0.8189 - auc: 0.8824 - loss: 0.8407 - val_Accuracy: 0.8082 - val_auc:
0.8723 - val_loss: 0.4376
Epoch 10/20
167/167            4s 23ms/step -
```

```
Accuracy: 0.8255 - auc: 0.8864 - loss: 0.8230 - val_Accuracy: 0.8065 - val_auc:
0.8729 - val_loss: 0.4346
Epoch 11/20
167/167          4s 24ms/step -
Accuracy: 0.8353 - auc: 0.8964 - loss: 0.7893 - val_Accuracy: 0.8060 - val_auc:
0.8733 - val_loss: 0.4388
Epoch 12/20
167/167          4s 23ms/step -
Accuracy: 0.8190 - auc: 0.8870 - loss: 0.8251 - val_Accuracy: 0.8117 - val_auc:
0.8732 - val_loss: 0.4329
Epoch 13/20
167/167          4s 23ms/step -
Accuracy: 0.8295 - auc: 0.8963 - loss: 0.7905 - val_Accuracy: 0.7968 - val_auc:
0.8732 - val_loss: 0.4462
Epoch 14/20
167/167          4s 23ms/step -
Accuracy: 0.8211 - auc: 0.8892 - loss: 0.8159 - val_Accuracy: 0.8100 - val_auc:
0.8733 - val_loss: 0.4352
Epoch 15/20
167/167          4s 23ms/step -
Accuracy: 0.8250 - auc: 0.8990 - loss: 0.7865 - val_Accuracy: 0.8065 - val_auc:
0.8735 - val_loss: 0.4326
Epoch 16/20
167/167          4s 23ms/step -
Accuracy: 0.8238 - auc: 0.8927 - loss: 0.7987 - val_Accuracy: 0.8052 - val_auc:
0.8734 - val_loss: 0.4330
Epoch 17/20
167/167          4s 23ms/step -
Accuracy: 0.8371 - auc: 0.9002 - loss: 0.7726 - val_Accuracy: 0.8065 - val_auc:
0.8738 - val_loss: 0.4317
Epoch 18/20
167/167          4s 23ms/step -
Accuracy: 0.8341 - auc: 0.8987 - loss: 0.7873 - val_Accuracy: 0.8087 - val_auc:
0.8742 - val_loss: 0.4338
Epoch 19/20
167/167          4s 22ms/step -
Accuracy: 0.8369 - auc: 0.9052 - loss: 0.7611 - val_Accuracy: 0.8078 - val_auc:
0.8738 - val_loss: 0.4383
Epoch 20/20
167/167          4s 24ms/step -
Accuracy: 0.8482 - auc: 0.9096 - loss: 0.7418 - val_Accuracy: 0.8039 - val_auc:
0.8706 - val_loss: 0.4405
```

[102]:
```
<keras.src.callbacks.history.History at 0x3173fc0a0>
```

[103]:
```
best_model = load_model('tf_model/model.keras')
```

```python
predictions = (best_model.predict(X_test_pad) > 0.5).astype(int)
predictions
output_df = pd.DataFrame(df_test['id'])
output_df['target'] = predictions
out = output_df.to_csv('Disaster_tweets_RNN.csv', index=False)
```

```
102/102              1s 6ms/step
```

The model got a score of 0.80416

[104]:
```python
model = Sequential([])

model.add(layers.Input(shape=(35,100)))
model.add(layers.LSTM(128, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.LSTM(128, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))

cp = ModelCheckpoint('tf_model/model_128.keras', save_best_only=True)

model.compile(optimizer=Adam(learning_rate=0.0001), loss=BinaryCrossentropy(), metrics=['Accuracy', AUC(name='auc')])

model.fit(X_train_pad, y_train, validation_data=(X_val_pad, y_val), epochs=50, callbacks=[cp], class_weight=weights)

best_model = load_model('tf_model/model_128.keras')

predictions = (best_model.predict(X_test_pad) > 0.5).astype(int)
predictions
output_df = pd.DataFrame(df_test['id'])
output_df['target'] = predictions
out = output_df.to_csv('Disaster_tweets_RNN_128.csv', index=False)
```

```
Epoch 1/50
167/167              10s 53ms/step -
Accuracy: 0.7043 - auc: 0.7413 - loss: 1.2223 - val_Accuracy: 0.7662 - val_auc:
0.8412 - val_loss: 0.4967
Epoch 2/50
167/167              10s 57ms/step -
Accuracy: 0.7967 - auc: 0.8566 - loss: 0.9263 - val_Accuracy: 0.7863 - val_auc:
0.8581 - val_loss: 0.4636
Epoch 3/50
167/167              10s 59ms/step -
Accuracy: 0.8006 - auc: 0.8637 - loss: 0.9001 - val_Accuracy: 0.7938 - val_auc:
```

```
0.8662 - val_loss: 0.4582
Epoch 4/50
167/167            10s 61ms/step -
Accuracy: 0.8125 - auc: 0.8760 - loss: 0.8635 - val_Accuracy: 0.8012 - val_auc:
0.8689 - val_loss: 0.4436
Epoch 5/50
167/167            10s 60ms/step -
Accuracy: 0.8211 - auc: 0.8859 - loss: 0.8327 - val_Accuracy: 0.8078 - val_auc:
0.8704 - val_loss: 0.4360
Epoch 6/50
167/167            8s 48ms/step -
Accuracy: 0.8167 - auc: 0.8854 - loss: 0.8302 - val_Accuracy: 0.8039 - val_auc:
0.8713 - val_loss: 0.4360
Epoch 7/50
167/167            8s 45ms/step -
Accuracy: 0.8097 - auc: 0.8774 - loss: 0.8599 - val_Accuracy: 0.8078 - val_auc:
0.8716 - val_loss: 0.4368
Epoch 8/50
167/167            8s 46ms/step -
Accuracy: 0.8120 - auc: 0.8809 - loss: 0.8489 - val_Accuracy: 0.8060 - val_auc:
0.8725 - val_loss: 0.4321
Epoch 9/50
167/167            8s 45ms/step -
Accuracy: 0.8274 - auc: 0.8978 - loss: 0.7882 - val_Accuracy: 0.8082 - val_auc:
0.8737 - val_loss: 0.4296
Epoch 10/50
167/167            7s 45ms/step -
Accuracy: 0.8253 - auc: 0.8935 - loss: 0.8098 - val_Accuracy: 0.8100 - val_auc:
0.8725 - val_loss: 0.4343
Epoch 11/50
167/167            8s 45ms/step -
Accuracy: 0.8265 - auc: 0.8918 - loss: 0.7994 - val_Accuracy: 0.8052 - val_auc:
0.8707 - val_loss: 0.4372
Epoch 12/50
167/167            7s 43ms/step -
Accuracy: 0.8293 - auc: 0.9039 - loss: 0.7630 - val_Accuracy: 0.8047 - val_auc:
0.8732 - val_loss: 0.4361
Epoch 13/50
167/167            8s 47ms/step -
Accuracy: 0.8367 - auc: 0.9059 - loss: 0.7608 - val_Accuracy: 0.8034 - val_auc:
0.8740 - val_loss: 0.4454
Epoch 14/50
167/167            8s 48ms/step -
Accuracy: 0.8363 - auc: 0.9072 - loss: 0.7464 - val_Accuracy: 0.8030 - val_auc:
0.8712 - val_loss: 0.4337
Epoch 15/50
167/167            8s 47ms/step -
Accuracy: 0.8443 - auc: 0.9086 - loss: 0.7425 - val_Accuracy: 0.8069 - val_auc:
```

```
0.8721 - val_loss: 0.4392
Epoch 16/50
167/167            8s 49ms/step -
Accuracy: 0.8388 - auc: 0.9080 - loss: 0.7453 - val_Accuracy: 0.8082 - val_auc:
0.8701 - val_loss: 0.4461
Epoch 17/50
167/167            8s 48ms/step -
Accuracy: 0.8499 - auc: 0.9181 - loss: 0.7088 - val_Accuracy: 0.7758 - val_auc:
0.8667 - val_loss: 0.5004
Epoch 18/50
167/167            8s 47ms/step -
Accuracy: 0.8458 - auc: 0.9195 - loss: 0.7044 - val_Accuracy: 0.7995 - val_auc:
0.8705 - val_loss: 0.4676
Epoch 19/50
167/167            8s 46ms/step -
Accuracy: 0.8538 - auc: 0.9239 - loss: 0.6787 - val_Accuracy: 0.8082 - val_auc:
0.8674 - val_loss: 0.4491
Epoch 20/50
167/167            8s 47ms/step -
Accuracy: 0.8633 - auc: 0.9296 - loss: 0.6547 - val_Accuracy: 0.7929 - val_auc:
0.8668 - val_loss: 0.4982
Epoch 21/50
167/167            8s 46ms/step -
Accuracy: 0.8676 - auc: 0.9341 - loss: 0.6358 - val_Accuracy: 0.8052 - val_auc:
0.8652 - val_loss: 0.4538
Epoch 22/50
167/167            8s 46ms/step -
Accuracy: 0.8629 - auc: 0.9317 - loss: 0.6540 - val_Accuracy: 0.8060 - val_auc:
0.8628 - val_loss: 0.4700
Epoch 23/50
167/167            8s 46ms/step -
Accuracy: 0.8727 - auc: 0.9389 - loss: 0.6127 - val_Accuracy: 0.7973 - val_auc:
0.8611 - val_loss: 0.5080
Epoch 24/50
167/167            8s 47ms/step -
Accuracy: 0.8736 - auc: 0.9429 - loss: 0.5980 - val_Accuracy: 0.8043 - val_auc:
0.8585 - val_loss: 0.4746
Epoch 25/50
167/167            8s 48ms/step -
Accuracy: 0.8803 - auc: 0.9515 - loss: 0.5528 - val_Accuracy: 0.7968 - val_auc:
0.8550 - val_loss: 0.5123
Epoch 26/50
167/167            8s 47ms/step -
Accuracy: 0.8896 - auc: 0.9557 - loss: 0.5266 - val_Accuracy: 0.7771 - val_auc:
0.8533 - val_loss: 0.5466
Epoch 27/50
167/167            8s 45ms/step -
Accuracy: 0.8983 - auc: 0.9563 - loss: 0.5196 - val_Accuracy: 0.7842 - val_auc:
```

```
0.8515 - val_loss: 0.5469
Epoch 28/50
167/167            8s 48ms/step -
Accuracy: 0.8944 - auc: 0.9614 - loss: 0.4960 - val_Accuracy: 0.7947 - val_auc:
0.8453 - val_loss: 0.5197
Epoch 29/50
167/167            8s 47ms/step -
Accuracy: 0.8933 - auc: 0.9619 - loss: 0.4933 - val_Accuracy: 0.7925 - val_auc:
0.8504 - val_loss: 0.5921
Epoch 30/50
167/167            8s 46ms/step -
Accuracy: 0.9062 - auc: 0.9662 - loss: 0.4565 - val_Accuracy: 0.7977 - val_auc:
0.8488 - val_loss: 0.5655
Epoch 31/50
167/167            8s 47ms/step -
Accuracy: 0.9024 - auc: 0.9633 - loss: 0.4761 - val_Accuracy: 0.7793 - val_auc:
0.8493 - val_loss: 0.6162
Epoch 32/50
167/167            8s 46ms/step -
Accuracy: 0.9111 - auc: 0.9707 - loss: 0.4249 - val_Accuracy: 0.7929 - val_auc:
0.8390 - val_loss: 0.6489
Epoch 33/50
167/167            8s 47ms/step -
Accuracy: 0.9128 - auc: 0.9725 - loss: 0.4147 - val_Accuracy: 0.7986 - val_auc:
0.8456 - val_loss: 0.6494
Epoch 34/50
167/167            8s 46ms/step -
Accuracy: 0.9267 - auc: 0.9788 - loss: 0.3686 - val_Accuracy: 0.7868 - val_auc:
0.8402 - val_loss: 0.7254
Epoch 35/50
167/167            8s 46ms/step -
Accuracy: 0.9224 - auc: 0.9752 - loss: 0.3915 - val_Accuracy: 0.7824 - val_auc:
0.8404 - val_loss: 0.6878
Epoch 36/50
167/167            8s 46ms/step -
Accuracy: 0.9308 - auc: 0.9818 - loss: 0.3408 - val_Accuracy: 0.7820 - val_auc:
0.8381 - val_loss: 0.7623
Epoch 37/50
167/167            8s 46ms/step -
Accuracy: 0.9274 - auc: 0.9821 - loss: 0.3325 - val_Accuracy: 0.7789 - val_auc:
0.8414 - val_loss: 0.7350
Epoch 38/50
167/167            8s 46ms/step -
Accuracy: 0.9313 - auc: 0.9817 - loss: 0.3361 - val_Accuracy: 0.7434 - val_auc:
0.8343 - val_loss: 0.7677
Epoch 39/50
167/167            8s 46ms/step -
Accuracy: 0.9313 - auc: 0.9834 - loss: 0.3292 - val_Accuracy: 0.7785 - val_auc:
```

```
0.8368 - val_loss: 0.7608
Epoch 40/50
167/167            8s 46ms/step -
Accuracy: 0.9378 - auc: 0.9834 - loss: 0.3243 - val_Accuracy: 0.7557 - val_auc:
0.8331 - val_loss: 0.8572
Epoch 41/50
167/167            8s 46ms/step -
Accuracy: 0.9417 - auc: 0.9885 - loss: 0.2727 - val_Accuracy: 0.7789 - val_auc:
0.8294 - val_loss: 0.8503
Epoch 42/50
167/167            8s 46ms/step -
Accuracy: 0.9475 - auc: 0.9892 - loss: 0.2613 - val_Accuracy: 0.7780 - val_auc:
0.8210 - val_loss: 0.8456
Epoch 43/50
167/167            8s 46ms/step -
Accuracy: 0.9474 - auc: 0.9882 - loss: 0.2720 - val_Accuracy: 0.7837 - val_auc:
0.8336 - val_loss: 0.8340
Epoch 44/50
167/167            8s 46ms/step -
Accuracy: 0.9407 - auc: 0.9850 - loss: 0.3032 - val_Accuracy: 0.7741 - val_auc:
0.8287 - val_loss: 0.9206
Epoch 45/50
167/167            8s 46ms/step -
Accuracy: 0.9542 - auc: 0.9919 - loss: 0.2310 - val_Accuracy: 0.7732 - val_auc:
0.8278 - val_loss: 0.8969
Epoch 46/50
167/167            8s 46ms/step -
Accuracy: 0.9535 - auc: 0.9901 - loss: 0.2515 - val_Accuracy: 0.7509 - val_auc:
0.8221 - val_loss: 0.9952
Epoch 47/50
167/167            8s 48ms/step -
Accuracy: 0.9437 - auc: 0.9880 - loss: 0.2827 - val_Accuracy: 0.7688 - val_auc:
0.8240 - val_loss: 0.9393
Epoch 48/50
167/167            8s 45ms/step -
Accuracy: 0.9599 - auc: 0.9928 - loss: 0.2146 - val_Accuracy: 0.7912 - val_auc:
0.8226 - val_loss: 0.9998
Epoch 49/50
167/167            8s 47ms/step -
Accuracy: 0.9612 - auc: 0.9941 - loss: 0.1952 - val_Accuracy: 0.7815 - val_auc:
0.8297 - val_loss: 0.9987
Epoch 50/50
167/167            8s 46ms/step -
Accuracy: 0.9569 - auc: 0.9931 - loss: 0.2100 - val_Accuracy: 0.7780 - val_auc:
0.8234 - val_loss: 0.9592
102/102            1s 12ms/step
```

This model with 128 units and 50 epochs doesnt perform any better than the model with 64 units and 20 epochs.

Disaster_tweets_RNN_128.csv 0.80140
Complete · 18s ago

Changing the learning rate to 0.00001 (not captured here) also didnt help with increasing the accuracy

Let us replace LSTM with GRU

[108]:
```
model = Sequential([])
model.add(layers.Input(shape=(35,100)))
model.add(layers.GRU(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.GRU(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))

cp = ModelCheckpoint('tf_model/model_gru.keras', save_best_only=True)

model.compile(optimizer=Adam(learning_rate=0.01), loss=BinaryCrossentropy(), metrics=['Accuracy'])

model.fit(X_train_pad, y_train, validation_data=(X_val_pad, y_val), epochs=30, callbacks=[cp], class_weight=weights, batch_size=512)

best_model = load_model('tf_model/model_gru.keras')

predictions = (best_model.predict(X_test_pad) > 0.5).astype(int)
predictions
output_df = pd.DataFrame(df_test['id'])
output_df['target'] = predictions
out = output_df.to_csv('Disaster_tweets_RNN_gru.csv', index=False)
```

```
Epoch 1/30
11/11            3s 191ms/step -
Accuracy: 0.5691 - loss: 1.4738 - val_Accuracy: 0.7680 - val_loss: 0.5020
Epoch 2/30
11/11            2s 185ms/step -
Accuracy: 0.7757 - loss: 0.9967 - val_Accuracy: 0.7802 - val_loss: 0.4829
Epoch 3/30
11/11            2s 191ms/step -
Accuracy: 0.8056 - loss: 0.8889 - val_Accuracy: 0.7881 - val_loss: 0.4583
Epoch 4/30
11/11            2s 190ms/step -
Accuracy: 0.8186 - loss: 0.8311 - val_Accuracy: 0.8069 - val_loss: 0.4358
Epoch 5/30
11/11            2s 194ms/step -
Accuracy: 0.8335 - loss: 0.7582 - val_Accuracy: 0.8122 - val_loss: 0.4369
```

```
Epoch 6/30
11/11            2s 197ms/step -
Accuracy: 0.8640 - loss: 0.6833 - val_Accuracy: 0.7846 - val_loss: 0.4692
Epoch 7/30
11/11            2s 193ms/step -
Accuracy: 0.8720 - loss: 0.6241 - val_Accuracy: 0.7947 - val_loss: 0.5146
Epoch 8/30
11/11            2s 198ms/step -
Accuracy: 0.8821 - loss: 0.5609 - val_Accuracy: 0.8012 - val_loss: 0.5165
Epoch 9/30
11/11            2s 200ms/step -
Accuracy: 0.8949 - loss: 0.4900 - val_Accuracy: 0.7693 - val_loss: 0.6139
Epoch 10/30
11/11            2s 188ms/step -
Accuracy: 0.9224 - loss: 0.3883 - val_Accuracy: 0.7806 - val_loss: 0.7008
Epoch 11/30
11/11            2s 202ms/step -
Accuracy: 0.9487 - loss: 0.2721 - val_Accuracy: 0.7758 - val_loss: 0.8283
Epoch 12/30
11/11            2s 193ms/step -
Accuracy: 0.9579 - loss: 0.2330 - val_Accuracy: 0.7793 - val_loss: 0.8612
Epoch 13/30
11/11            2s 188ms/step -
Accuracy: 0.9465 - loss: 0.2846 - val_Accuracy: 0.7614 - val_loss: 0.7830
Epoch 14/30
11/11            2s 185ms/step -
Accuracy: 0.9607 - loss: 0.2194 - val_Accuracy: 0.7680 - val_loss: 0.9275
Epoch 15/30
11/11            2s 191ms/step -
Accuracy: 0.9682 - loss: 0.1822 - val_Accuracy: 0.7785 - val_loss: 0.8825
Epoch 16/30
11/11            2s 197ms/step -
Accuracy: 0.9780 - loss: 0.1417 - val_Accuracy: 0.7785 - val_loss: 1.0798
Epoch 17/30
11/11            2s 204ms/step -
Accuracy: 0.9803 - loss: 0.1145 - val_Accuracy: 0.7763 - val_loss: 1.0643
Epoch 18/30
11/11            2s 194ms/step -
Accuracy: 0.9802 - loss: 0.1304 - val_Accuracy: 0.7837 - val_loss: 1.0214
Epoch 19/30
11/11            2s 196ms/step -
Accuracy: 0.9779 - loss: 0.1360 - val_Accuracy: 0.7780 - val_loss: 1.0073
Epoch 20/30
11/11            2s 192ms/step -
Accuracy: 0.9854 - loss: 0.1095 - val_Accuracy: 0.7771 - val_loss: 1.0967
Epoch 21/30
11/11            2s 186ms/step -
Accuracy: 0.9786 - loss: 0.1163 - val_Accuracy: 0.7758 - val_loss: 1.1032
```

```
Epoch 22/30
11/11          2s 192ms/step -
Accuracy: 0.9839 - loss: 0.0935 - val_Accuracy: 0.7697 - val_loss: 1.0899
Epoch 23/30
11/11          2s 187ms/step -
Accuracy: 0.9827 - loss: 0.0883 - val_Accuracy: 0.7644 - val_loss: 1.0857
Epoch 24/30
11/11          2s 197ms/step -
Accuracy: 0.9831 - loss: 0.0804 - val_Accuracy: 0.7701 - val_loss: 1.2210
Epoch 25/30
11/11          2s 196ms/step -
Accuracy: 0.9833 - loss: 0.0944 - val_Accuracy: 0.7583 - val_loss: 1.0229
Epoch 26/30
11/11          2s 199ms/step -
Accuracy: 0.9845 - loss: 0.0820 - val_Accuracy: 0.7758 - val_loss: 1.2088
Epoch 27/30
11/11          2s 193ms/step -
Accuracy: 0.9792 - loss: 0.1074 - val_Accuracy: 0.7815 - val_loss: 1.0998
Epoch 28/30
11/11          2s 202ms/step -
Accuracy: 0.9852 - loss: 0.0779 - val_Accuracy: 0.7732 - val_loss: 1.2302
Epoch 29/30
11/11          2s 195ms/step -
Accuracy: 0.9834 - loss: 0.0764 - val_Accuracy: 0.7715 - val_loss: 1.2893
Epoch 30/30
11/11          2s 193ms/step -
Accuracy: 0.9859 - loss: 0.0661 - val_Accuracy: 0.7728 - val_loss: 1.3217
102/102          1s 5ms/step
```

With learning rate reduced to 0.01, replacing LSTM with GRU and validation to just Accuracy, batch size at 512 and epochs at 30, the model is slightly better at 0.81152

0.5 Step 4: Results and Analysis

[23]:
```python
from tensorflow.keras.callbacks import TensorBoard

model = Sequential([])
model.add(layers.Input(shape=(35,100)))
model.add(layers.GRU(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.GRU(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))

log_dir = 'tf_model_gru'
```

```
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)

model.compile(optimizer=Adam(learning_rate=0.01), loss=BinaryCrossentropy(),␣
 ↪metrics=['Accuracy'])

model.fit(X_train_pad, y_train, validation_data=(X_val_pad, y_val), epochs=30,␣
 ↪callbacks=[tensorboard_callback], class_weight=weights, batch_size=512)
```

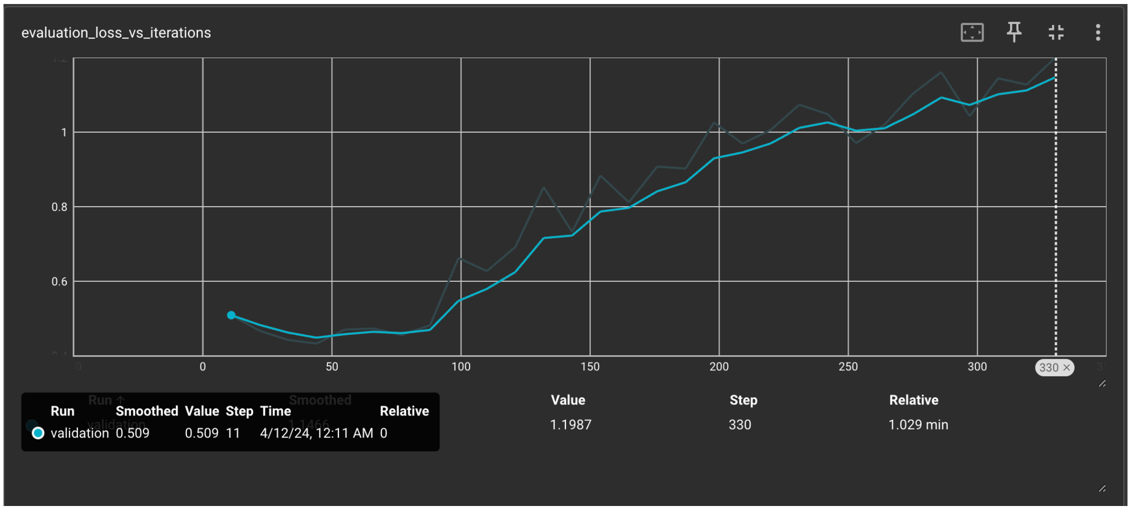
```
Epoch 1/30
11/11                3s 181ms/step -
Accuracy: 0.5639 - loss: 1.5175 - val_Accuracy: 0.7566 - val_loss: 0.5090
Epoch 2/30
11/11                2s 180ms/step -
Accuracy: 0.7842 - loss: 0.9594 - val_Accuracy: 0.7833 - val_loss: 0.4672
Epoch 3/30
11/11                2s 189ms/step -
Accuracy: 0.7933 - loss: 0.9042 - val_Accuracy: 0.8104 - val_loss: 0.4425
Epoch 4/30
11/11                2s 187ms/step -
Accuracy: 0.8169 - loss: 0.8274 - val_Accuracy: 0.8122 - val_loss: 0.4328
Epoch 5/30
11/11                2s 190ms/step -
Accuracy: 0.8281 - loss: 0.7867 - val_Accuracy: 0.7885 - val_loss: 0.4702
Epoch 6/30
11/11                2s 187ms/step -
Accuracy: 0.8466 - loss: 0.7219 - val_Accuracy: 0.7890 - val_loss: 0.4732
Epoch 7/30
11/11                2s 187ms/step -
Accuracy: 0.8520 - loss: 0.6713 - val_Accuracy: 0.7999 - val_loss: 0.4557
Epoch 8/30
11/11                2s 193ms/step -
Accuracy: 0.8825 - loss: 0.6105 - val_Accuracy: 0.7920 - val_loss: 0.4821
Epoch 9/30
11/11                2s 190ms/step -
Accuracy: 0.9089 - loss: 0.4950 - val_Accuracy: 0.7824 - val_loss: 0.6624
Epoch 10/30
11/11                2s 194ms/step -
Accuracy: 0.9150 - loss: 0.4480 - val_Accuracy: 0.7868 - val_loss: 0.6272
Epoch 11/30
11/11                2s 191ms/step -
Accuracy: 0.9420 - loss: 0.3299 - val_Accuracy: 0.7890 - val_loss: 0.6917
Epoch 12/30
11/11                2s 202ms/step -
Accuracy: 0.9559 - loss: 0.2587 - val_Accuracy: 0.7601 - val_loss: 0.8525
Epoch 13/30
11/11                2s 195ms/step -
Accuracy: 0.9517 - loss: 0.2490 - val_Accuracy: 0.7771 - val_loss: 0.7325
Epoch 14/30
```

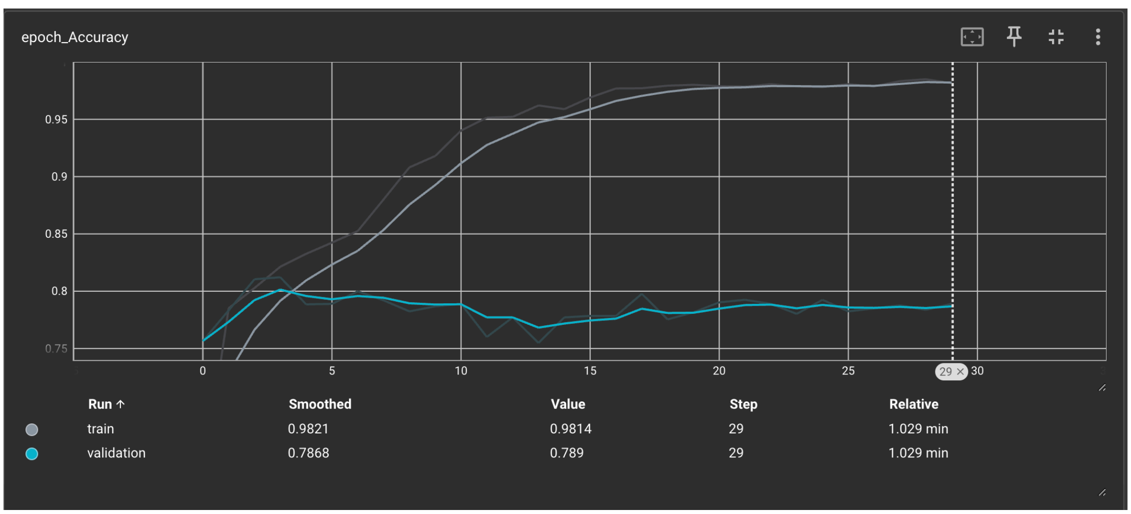
```
11/11            2s 195ms/step -
Accuracy: 0.9666 - loss: 0.1905 - val_Accuracy: 0.7548 - val_loss: 0.8835
Epoch 15/30
11/11            2s 191ms/step -
Accuracy: 0.9602 - loss: 0.2172 - val_Accuracy: 0.7771 - val_loss: 0.8118
Epoch 16/30
11/11            2s 189ms/step -
Accuracy: 0.9693 - loss: 0.1879 - val_Accuracy: 0.7785 - val_loss: 0.9077
Epoch 17/30
11/11            2s 192ms/step -
Accuracy: 0.9811 - loss: 0.1292 - val_Accuracy: 0.7785 - val_loss: 0.9023
Epoch 18/30
11/11            2s 195ms/step -
Accuracy: 0.9790 - loss: 0.1280 - val_Accuracy: 0.7977 - val_loss: 1.0260
Epoch 19/30
11/11            2s 195ms/step -
Accuracy: 0.9816 - loss: 0.1095 - val_Accuracy: 0.7754 - val_loss: 0.9692
Epoch 20/30
11/11            2s 202ms/step -
Accuracy: 0.9801 - loss: 0.1165 - val_Accuracy: 0.7815 - val_loss: 1.0063
Epoch 21/30
11/11            2s 193ms/step -
Accuracy: 0.9822 - loss: 0.0910 - val_Accuracy: 0.7903 - val_loss: 1.0736
Epoch 22/30
11/11            2s 196ms/step -
Accuracy: 0.9813 - loss: 0.1086 - val_Accuracy: 0.7925 - val_loss: 1.0481
Epoch 23/30
11/11            2s 206ms/step -
Accuracy: 0.9819 - loss: 0.1089 - val_Accuracy: 0.7890 - val_loss: 0.9706
Epoch 24/30
11/11            2s 206ms/step -
Accuracy: 0.9800 - loss: 0.1036 - val_Accuracy: 0.7802 - val_loss: 1.0200
Epoch 25/30
11/11            2s 209ms/step -
Accuracy: 0.9810 - loss: 0.0941 - val_Accuracy: 0.7925 - val_loss: 1.1031
Epoch 26/30
11/11            2s 197ms/step -
Accuracy: 0.9823 - loss: 0.0886 - val_Accuracy: 0.7824 - val_loss: 1.1613
Epoch 27/30
11/11            2s 190ms/step -
Accuracy: 0.9759 - loss: 0.0970 - val_Accuracy: 0.7850 - val_loss: 1.0427
Epoch 28/30
11/11            2s 198ms/step -
Accuracy: 0.9862 - loss: 0.0787 - val_Accuracy: 0.7877 - val_loss: 1.1444
Epoch 29/30
11/11            2s 211ms/step -
Accuracy: 0.9861 - loss: 0.0759 - val_Accuracy: 0.7837 - val_loss: 1.1274
Epoch 30/30
```

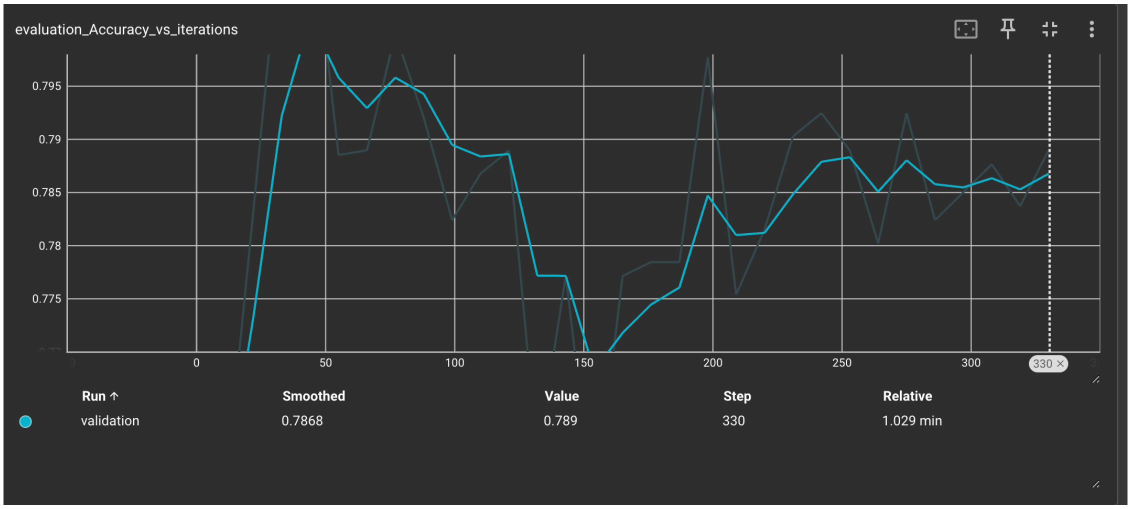
```
11/11                 2s 206ms/step -
Accuracy: 0.9809 - loss: 0.0842 - val_Accuracy: 0.7890 - val_loss: 1.1987
```

[23]:
```
<keras.src.callbacks.history.History at 0x3135b8c40>
```

TensorBoard for the GRU model:







We have explored LSTM and GRU based RNN. For the models proposed above, the models have returned results between 0.79-0.81. Due to lack of time, and since the primary expectation from this project is not accuracy, I am settling for these values.

A brief overview of what was attempted: | RNN family | units | learning rate| epochs | weights | batch size | metrics | test result | |:——–:|:——–:|:——–:|:——–:|:——–:|:——–:|:——–:|:——–:| | LSTM | 64 | 0.0001 | 20 | No | default | [‘Accuracy’, AUC(name=‘auc’)] | 0.79773 | | LSTM | 64 | 0.0001 | 20 | Yes | default | [‘Accuracy’, AUC(name=‘auc’)] | 0.80416 | | LSTM | 128 | 0.0001 | 50 | Yes | default | [‘Accuracy’, AUC(name=‘auc’)] | 0.80140 | | GRU | 64 | 0.01 | 30 | Yes | 512 | [‘Accuracy’] | 0.81152 |

metrics ‘Accuracy’ works well enough for most cases as has been seen in the results. I have explored fine tuning hyper-parameters and other values to see how the model reacts for both validation as well as test data. The values I used for learning rate or epoch does play a role in the accuracy but the results were not with a big margin.

I haven’t shown all the models I tried since a few of the tweaks in hyper parameters didn’t yield better results. Some of those models performed worse than the above models.

Decreasing the learning rate further to 0.00001 didn’t help with the accuracy. epochs at 20 or 50 are also not really much helpful in these models. I could have probably gotten similar results with a lesser epoch.

0.6 Step 5: Conclusion

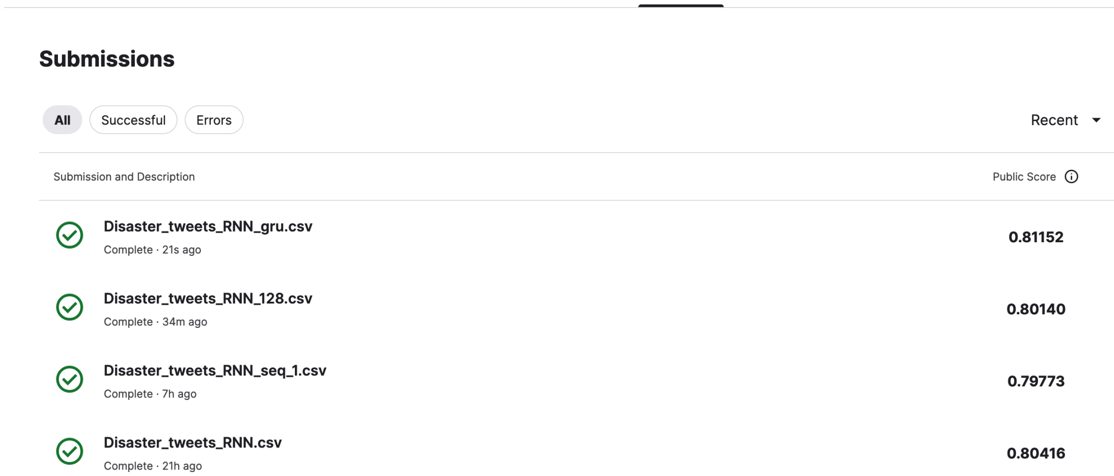
Results and key learnings The key learning is that tweaking hyperparameter alone is not enough in many cases. We need to fine tune the model based on the kind of data that is being used in the models. We could clean up the data a little more including removal of http links and explore further stop words removal. While the results are decent, unfortunately I would have been happy if I had come up with a model with a result > 0.9

Try in future I have used only Adam optimizer and would like to try other optimizers to see how they perform for this use case. I have used GloVe embeddings and would try other word embeddings

0.7 Step 6: Produce Deliverables

GitHub Repository: https://github.com/krishnakuruvadi/Week4_RNN

Report: https://github.com/krishnakuruvadi/Week4_RNN



Kaggle leaderboard:

[ ]: