DTSA-5511 Week 4 Kaggle Project

April 28, 2024

1 Brief description of the problem and data

The purpose of this project is to classify tweet data as one that is related to natural disaster information or not using a neural network model. Let's take a look at what the data looks like, its shape, balance, etc.

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

[103]:
```
train = pd.read_csv('C:/School/Machine Learning/5511 HW/Week 4/train.csv')
test = pd.read_csv('C:/School/Machine Learning/5511 HW/Week 4/test.csv')
```

[104]:
```
train.head()
```

[104]:
```
   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 …

   target
0       1
1       1
2       1
3       1
4       1
```

[105]:
```
test.head()
```

[105]:
```
   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
```

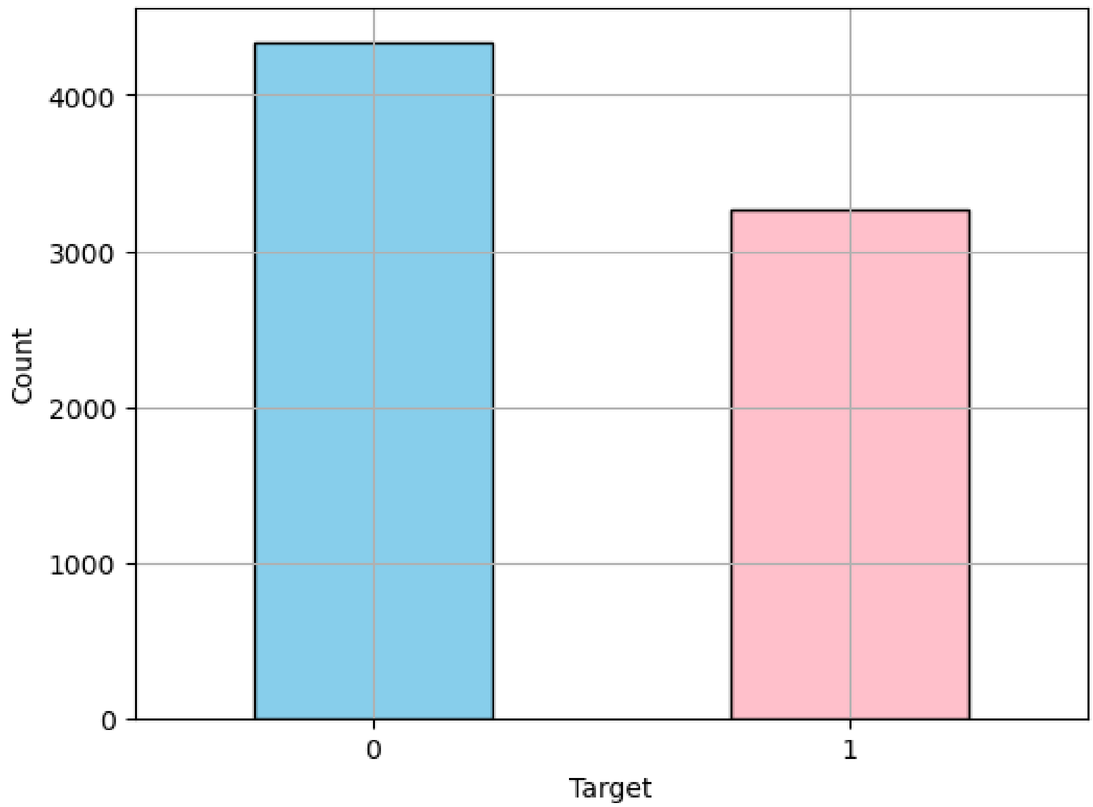
[106]:
```
train.describe()
```

[106]:
```
                 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
```

[107]:
```
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
```

[108]:
```
train['target'].value_counts().plot(kind='bar', color=['skyblue', 'pink'],␣
 ↪edgecolor='black')
plt.xlabel('Target')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.grid()
plt.show()
```



So as we can see we have 7613 tweets in our training data and the data is fairly balanced. We do have some NaNs we are going to have to deal with.

2 EDA

Let's look at what the tweets look like based on whether they are related to natural disaster or not. Let's also check for duplicates and drop those duplicate tweets.

[110]:
```
print(train[train['target'] == 0]['text'].head(10))
print(train[train['target'] == 1]['text'].head(10))
```

```
15                  What's up man?
16                   I love fruits
17                Summer is lovely
18               My car is so fast
19    What a goooooooaaaaaal!!!!!!
20          this is ridiculous…
21              London is cool ;)
22                     Love skiing
23           What a wonderful day!
24                        LOOOOOOL
Name: text, dtype: object
```

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

[111]:

```python
duplicate_tweets = train[train.duplicated(subset=['text'], keep=False)]
print(duplicate_tweets)
```

```
         id     keyword        location  \
40       59      ablaze  Live On Webcam
48       68      ablaze  Live On Webcam
106     156  aftershock              US
115     165  aftershock              US
118     171  aftershock     Switzerland
…       …           …             …
7600  10855         NaN             NaN
7607  10867         NaN             NaN
7609  10870         NaN             NaN
7610  10871         NaN             NaN
7611  10872         NaN             NaN

                                                   text  target
40    Check these out: http://t.co/rOI2NSmEJJ http:/…       0
48    Check these out: http://t.co/rOI2NSmEJJ http:/…       0
106   320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/vA…       0
115   320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/vA…       0
118   320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/TH…       0
…                                                 …     …
7600  Evacuation order lifted for town of Roosevelt:…       1
7607  #stormchase Violent Record Breaking EF-5 El Re…       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

[179 rows x 5 columns]
```

[112]:

```python
train.drop(duplicate_tweets.index, inplace=True)
```

Now we have to process the text data. I used re and nltk for word processing, I also changed all the contractions to full words, I used urllib to parse URLs, as well as spellchecker to correct any misspellings which are likely to occur in tweets. I got rid of all symbols and emojis and lemmatized the words for easier processing.

[113]:

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from urllib.parse import urlparse
from spellchecker import SpellChecker
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

class text_preprocess:
    def __init__(self):
        self.stop_words = set(stopwords.words('english'))
        self.lemmatizer = WordNetLemmatizer()
        self.spell = SpellChecker()
        self.contractions = {
            "ain't": "is not",
            "aren't": "are not",
            "can't": "cannot",
            "can't've": "cannot have",
            "could've": "could have",
            "couldn't": "could not",
            "didn't": "did not",
            "doesn't": "does not",
            "don't": "do not",
            "hadn't": "had not",
            "hasn't": "has not",
            "haven't": "have not",
            "he'd": "he would",
            "he'll": "he will",
            "he's": "he is",
            "how'd": "how did",
            "how'll": "how will",
            "how's": "how is",
            "i'd": "i would",
            "i'll": "i will",
            "i'm": "i am",
            "i've": "i have",
            "isn't": "is not",
            "it'd": "it would",
            "it'll": "it will",
            "it's": "it is",
            "let's": "let us",
            "ma'am": "madam",
            "mayn't": "may not",
            "might've": "might have",
```

```
            "mightn't": "might not",
            "must've": "must have",
            "mustn't": "must not",
            "needn't": "need not",
            "oughtn't": "ought not",
            "shan't": "shall not",
            "sha'n't": "shall not",
            "she'd": "she would",
            "she'll": "she will",
            "she's": "she is",
            "should've": "should have",
            "shouldn't": "should not",
            "that'd": "that would",
            "that's": "that is",
            "there'd": "there had",
            "there's": "there is",
            "they'd": "they would",
            "they'll": "they will",
            "they're": "they are",
            "they've": "they have",
            "wasn't": "was not",
            "we'd": "we would",
            "we'll": "we will",
            "we're": "we are",
            "we've": "we have",
            "weren't": "were not",
            "what'll": "what will",
            "what're": "what are",
            "what's": "what is",
            "what've": "what have",
            "when's": "when is",
            "where'd": "where did",
            "where's": "where is",
            "who'll": "who will",
            "who's": "who is",
            "won't": "will not",
            "wouldn't": "would not",
            "you'd": "you would",
            "you'll": "you will",
            "you're": "you are",
            "you've": "you have"
        }

    def expand_contractions(self, text):
        pattern = re.compile(r'\b(' + '|'.join(self.contractions.keys()) + 
↪r')\b')
        def expand_match(contraction):
```

```
            match = contraction.group(0)
            return self.contractions.get(match)
        return pattern.sub(expand_match, text)

    def clean_text(self, text):
        text = self.expand_contractions(text)
        text = re.sub(r'[^\w\s]', '', text)
        tokens = word_tokenize(text)
        if tokens != None:
            tokens = [token for token in tokens if token.isalnum()]
            return tokens
        else:
            return 1
        urls = re.findall(r'http\S+', text)
        for url in urls:
            domain = urlparse(url).netloc
            text = text.replace(url, domain)
        text = text.encode('ascii', 'ignore').decode('ascii')
        tokens = [self.spell.correction(word) for word in tokens]
        lemmatizer = WordNetLemmatizer()
        stop_words = set(stopwords.words('english'))
        tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in␣
 ↪stop_words]
        return ' '.join(tokens)

preprocessor = text_preprocess()
train['text'] = train['text'].apply(preprocessor.expand_contractions)
train['text'] = train['text'].apply(preprocessor.clean_text)
test['text'] = test['text'].apply(preprocessor.expand_contractions)
test['text'] = test['text'].apply(preprocessor.clean_text)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Maciej\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Maciej\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Maciej\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[114]:
```
train.head(15)
```

[114]:

```
   id keyword location                                               text  \
0   1     NaN      NaN  [Our, Deeds, are, the, Reason, of, this, earth…
1   4     NaN      NaN      [Forest, fire, near, La, Ronge, Sask, Canada]
2   5     NaN      NaN  [All, residents, asked, to, shelter, in, place…
```

```
3    6      NaN       NaN  [13000, people, receive, wildfires, evacuation…
4    7      NaN       NaN  [Just, got, sent, this, photo, from, Ruby, Ala…
5    8      NaN       NaN  [RockyFire, Update, California, Hwy, 20, close…
6   10      NaN       NaN  [flood, disaster, Heavy, rain, causes, flash, …
7   13      NaN       NaN  [Im, on, top, of, the, hill, and, I, can, see,…
8   14      NaN       NaN  [Theres, an, emergency, evacuation, happening,…
9   15      NaN       NaN  [Im, afraid, that, the, tornado, is, coming, t…
10  16      NaN       NaN  [Three, people, died, from, the, heat, wave, s…
11  17      NaN       NaN  [Haha, South, Tampa, is, getting, flooded, hah…
12  18      NaN       NaN  [raining, flooding, Florida, TampaBay, Tampa, …
13  19      NaN       NaN      [Flood, in, Bago, Myanmar, We, arrived, Bago]
14  20      NaN       NaN  [Damage, to, school, bus, on, 80, in, multi, c…

    target
0        1
1        1
2        1
3        1
4        1
5        1
6        1
7        1
8        1
9        1
10       1
11       1
12       1
13       1
14       1
```

This is what our text data looks like now.

3 Model Architecture

The next step was processing the data in a way that can be interpreted by our model. First I wanted to use the keyword and location columns that are provided for us in the data, but since those need to be processed as well the simplest solution was to combine them with the text column. Then I split the train data. Since our preprocessed text needs to be interpretable for our sequential model we still need to do some processing. For that we used the keras libraries tokenizer and padder since were using the keras sequential model and it is easy to use and compatible.

[115]:
```
from sklearn.model_selection import train_test_split

train['text'] = train['text'].astype(str)
train['keyword'] = train['keyword'].fillna('').astype(str)
train['location'] = train['location'].fillna('').astype(str)
X = train['text'] + ' ' + train['keyword'] + ' ' + train['location']
```

```
y = train['target']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[116]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

train_sequences = tokenizer.texts_to_sequences(X_train)
val_sequences = tokenizer.texts_to_sequences(X_val)

X_train_padded = pad_sequences(train_sequences, maxlen=100)
X_val_padded = pad_sequences(val_sequences, maxlen=100)

vocab_size = len(tokenizer.word_index) + 1
```

Now our data is ready to be applied to the model. Our model consists of 4 layers. The first layer is the embedding layer that converts our processed text into vectors. Our second layer is the LSTM layer that processes the data while being able to store the information throughout the process. The third layer is the dropout layer that helps prevent overfitting. The final dense layer is a sigmoid activation function that determines the output.

[164]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dropout, Dense
from tensorflow.keras.optimizers import Adam

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=100))
model.add(LSTM(64))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])
```

4 Results and Analysis

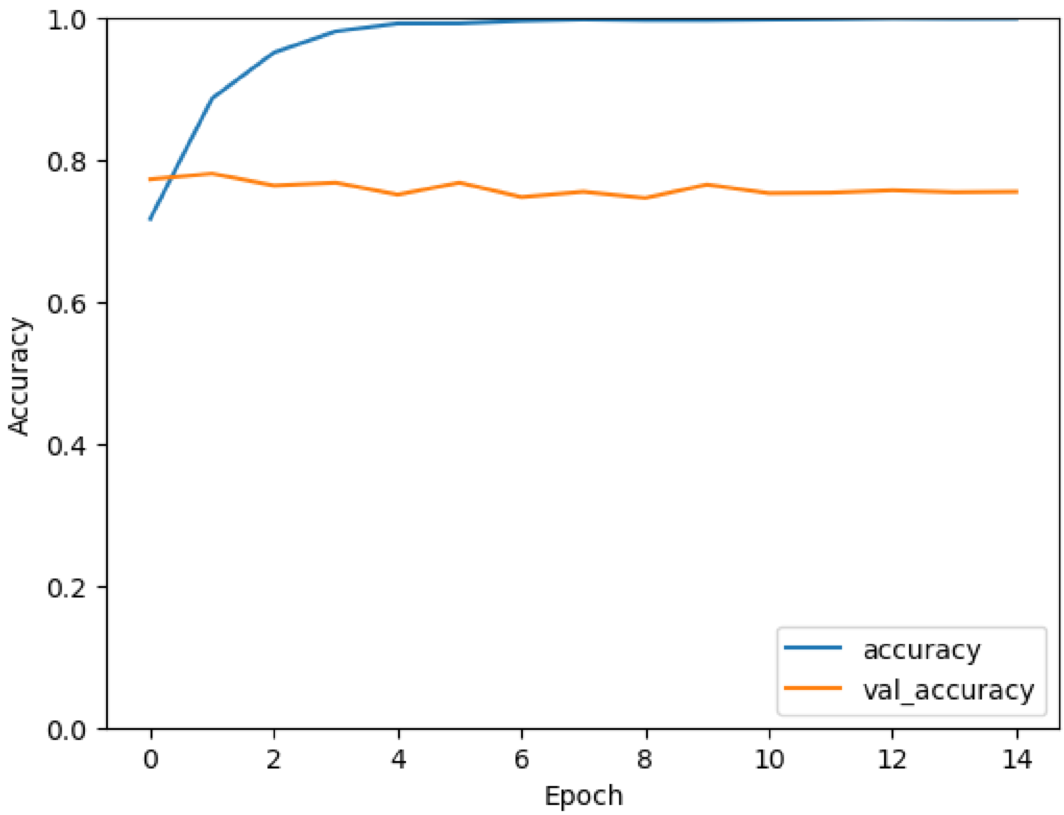
[165]:
```
history = model.fit(X_train_padded, y_train, epochs=15, batch_size=32,␣
↪validation_data=(X_val_padded, y_val))
```

```
Epoch 1/15
186/186              5s 23ms/step -
accuracy: 0.6489 - loss: 0.6211 - val_accuracy: 0.7727 - val_loss: 0.4829
Epoch 2/15
186/186              4s 22ms/step -
```

```
accuracy: 0.8830 - loss: 0.2994 - val_accuracy: 0.7808 - val_loss: 0.5038
Epoch 3/15
186/186           4s 24ms/step -
accuracy: 0.9577 - loss: 0.1275 - val_accuracy: 0.7640 - val_loss: 0.5692
Epoch 4/15
186/186           4s 23ms/step -
accuracy: 0.9809 - loss: 0.0645 - val_accuracy: 0.7680 - val_loss: 0.7380
Epoch 5/15
186/186           4s 24ms/step -
accuracy: 0.9947 - loss: 0.0244 - val_accuracy: 0.7512 - val_loss: 0.8443
Epoch 6/15
186/186           4s 23ms/step -
accuracy: 0.9954 - loss: 0.0146 - val_accuracy: 0.7680 - val_loss: 0.8677
Epoch 7/15
186/186           4s 24ms/step -
accuracy: 0.9965 - loss: 0.0136 - val_accuracy: 0.7478 - val_loss: 1.0594
Epoch 8/15
186/186           4s 24ms/step -
accuracy: 0.9983 - loss: 0.0067 - val_accuracy: 0.7552 - val_loss: 1.3343
Epoch 9/15
186/186           4s 22ms/step -
accuracy: 0.9971 - loss: 0.0073 - val_accuracy: 0.7465 - val_loss: 1.0794
Epoch 10/15
186/186           4s 23ms/step -
accuracy: 0.9977 - loss: 0.0072 - val_accuracy: 0.7653 - val_loss: 1.2752
Epoch 11/15
186/186           4s 24ms/step -
accuracy: 0.9981 - loss: 0.0065 - val_accuracy: 0.7532 - val_loss: 1.2565
Epoch 12/15
186/186           4s 23ms/step -
accuracy: 0.9983 - loss: 0.0044 - val_accuracy: 0.7539 - val_loss: 1.4297
Epoch 13/15
186/186           4s 22ms/step -
accuracy: 0.9997 - loss: 0.0018 - val_accuracy: 0.7572 - val_loss: 1.5471
Epoch 14/15
186/186           4s 22ms/step -
accuracy: 0.9989 - loss: 0.0019 - val_accuracy: 0.7545 - val_loss: 1.6403
Epoch 15/15
186/186           4s 22ms/step -
accuracy: 0.9996 - loss: 0.0012 - val_accuracy: 0.7552 - val_loss: 1.4803
```

[166]:
```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
```

```
plt.show()
```



So as you can see while the val_accuracy is reasonable, the val_loss increases quite quickly. That means the model is overfitting very fast. Let's try to optimize the hyperparameters to prevent this.

[19]:
```
from tensorflow.keras.callbacks import EarlyStopping
from keras_tuner import RandomSearch

def build_model(hp):
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=hp.
 ↪Int('embedding_units', min_value=50, max_value=200, step=50)))
    model.add(LSTM(units=hp.Int('lstm_units', min_value=32, max_value=128,␣
 ↪step=32)))
    model.add(Dropout(rate=hp.Float('dropout_rate', min_value=0.2, max_value=0.
 ↪5, step=0.1)))
    model.add(Dense(units=1, activation='sigmoid'))
    batch_size_hp = hp.Choice('batch_size', values=[8, 16, 32])
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
    return model
```

```
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=30,
    directory='hps',
    project_name='tweet classification'
)

early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)
tuner.search(X_train_padded, y_train, epochs=15, validation_data=(X_val_padded,␣
 ↪y_val), callbacks=[early_stopping])

best_params = tuner.get_best_hyperparameters(num_trials=1)[0]

final_model = tuner.hypermodel.build(best_params)
```

```
Trial 30 Complete [00h 00m 26s]
val_accuracy: 0.7989240288734436

Best val_accuracy So Far: 0.8184263706207275
Total elapsed time: 00h 19m 01s
```

[167]:
```
print(best_params.values)
```

```
{'embedding_units': 150, 'lstm_units': 32, 'dropout_rate': 0.4, 'batch_size':
32}
```

The best val_accuracy score achieved was 0.818 which is great, but unfortunately the val_loss was increasing really quickly with the number of epochs using the hyperparameters found by the tuner. I tried to nullify this but optimizing the hyperparameters by hand and trying different values to try to combat the problem. After spending a lot of time on this I was not able to figure out a solution, and with a low number of epochs and certain hyperparameters we still get fairly good results so I just stuck with my best results.

[173]:
```
from tensorflow.keras.optimizers import Adam
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=150))
model.add(LSTM(64))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
optimizer = Adam(learning_rate=0.00001)
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
history = model.fit(X_train_padded, y_train, epochs=4, batch_size=64,␣
 ↪validation_data=(X_val_padded, y_val))
```

```
Epoch 1/4
```

```
93/93              4s 31ms/step -
accuracy: 0.6263 - loss: 0.6425 - val_accuracy: 0.8063 - val_loss: 0.4461
Epoch 2/4
93/93              3s 30ms/step -
accuracy: 0.8804 - loss: 0.2947 - val_accuracy: 0.8016 - val_loss: 0.4626
Epoch 3/4
93/93              3s 28ms/step -
accuracy: 0.9524 - loss: 0.1346 - val_accuracy: 0.7814 - val_loss: 0.6450
Epoch 4/4
93/93              3s 28ms/step -
accuracy: 0.9865 - loss: 0.0519 - val_accuracy: 0.7747 - val_loss: 0.6794
```

This part is just applying our model to the test data and making our submission file for the Kaggle competition.

[87]:
```
test['text'] = test['text'].astype(str)
test['keyword'] = test['keyword'].fillna('').astype(str)
test['location'] = test['location'].fillna('').astype(str)
test_id = test['id']
test_combined = test['text'] + ' ' + test['keyword'] + ' ' + test['location']

tokenizer = Tokenizer()
tokenizer.fit_on_texts(test_combined)

test_sequences = tokenizer.texts_to_sequences(test_combined)

test_padded = pad_sequences(test_sequences, maxlen=100)
```

[162]:
```
y_pred = model.predict(test_padded)
y_pred = np.round(y_pred).astype(int).reshape(3263)

submission = pd.DataFrame({'id': test_id, 'target': y_pred})
submission.to_csv('submission.csv', index=False)
```

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

[163]:
```
submission.head(5)
```

[163]:

	id	target
0	0	0
1	2	0
2	3	0
3	9	0
4	11	1

5 Conclusion

In conclusion this project used a neural network model to classify twitter posts as related to natural disasters or unrelated. I was able to achieve a solid validation accuracy but I had problems with the model overfitting resulting in a not so great submission score. I tried to optimize the hyperparameters but unfortunately I ran into the same problem. After much research and trial and error, I'm still not sure what the problem is. It could be that I am adjusting the wrong hyperparameters or testing the wrong values for the hyperparameters. It could also be a problem with the design of my model, or a mistake I made in preprocessing. I tried to make sure those were not the issue but in the end I could not figure out what the actual issue was. If I had another chance to do this project I would try to reference someone else's work on Kaggle a bit more instead of trying to do something of my own. Overall my model still did okay but if I had more time I would definitely try to re-do the whole thing.