Natural Language Processing on Disaster Tweets

By *Alex Vaillant*

Dataset Source: https://www.kaggle.com/c/nlp-getting-started/overview

Research Question

To what extent can we predict if a tweet is about a Disaster using Natural Language Processing, Neural Networks, and Universal Sentence Encoder?

Set Up Environment

Import Necessary Libraries

In [1]:
```
import pandas as pd
import numpy as np
from numpy import random
import tensorflow as tf
import keras
import tensorflow_hub
import tensorflow_text
import seaborn as sns
from matplotlib import pyplot as plt
from keras import layers, Sequential
from keras.layers import Dense, Dropout
from keras.models import load_model
from keras.callbacks import EarlyStopping
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from platform import python_version
```

In [2]:
```
print(python_version())
```

```
3.7.10
```

In [3]:
```
RandomSeed = random.seed(94)
```

Load Universal Sentence Encoder via tensowflow_hub.load()

In [4]:
```
UniversalSentenceEncoder = tensorflow_hub.load("https://tfhub.dev/google/universal-sent
```

Data Gathering

In [5]:
```
DisasterTweetsUrl = "C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Sen
TweetTestData = "C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Sentime
RawDisasterTweetData = pd.read_csv(DisasterTweetsUrl)
TweetTestData = pd.read_csv(TweetTestData)
```

Exploratory Data Analysis

In [6]:
```
print("Training Data Shape:", RawDisasterTweetData.shape)
print("Training Data Columns:", RawDisasterTweetData.columns, "\n")
print("Testing Data Shape:", TweetTestData.shape)
print("Testing Data Columns:",TweetTestData.columns)
```

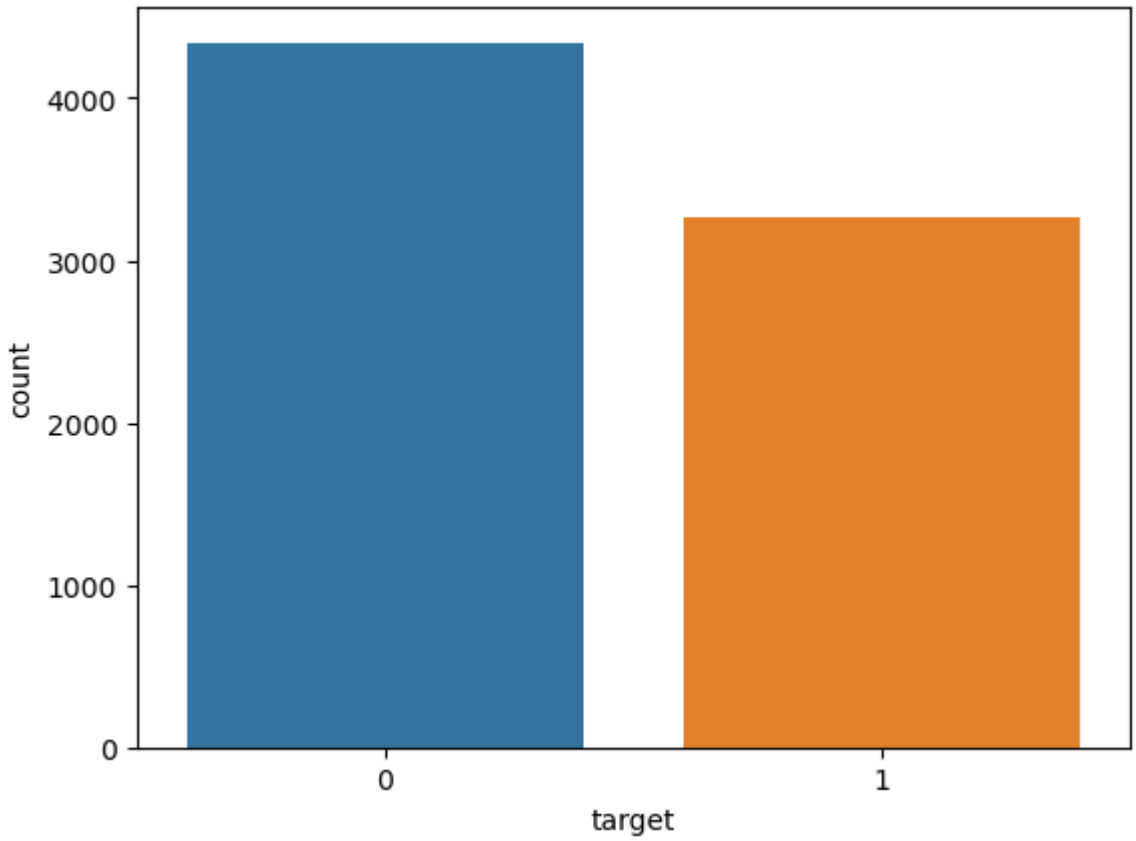
```
Training Data Shape: (7613, 5)
Training Data Columns: Index(['id', 'keyword', 'location', 'text', 'target'], dtype='obj
ect')

Testing Data Shape: (3263, 4)
Testing Data Columns: Index(['id', 'keyword', 'location', 'text'], dtype='object')
```

In [7]:
```
# Check data sparsity in the necessary fields (text and target)
RawDisasterTweetData.isnull().sum()
```

Out[7]:
```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

In [8]:
```
sns.countplot(x = 'target', data = RawDisasterTweetData)
plt.show()
```



There's an uneven distribution of 0 vs 1. Use Random Sampling to fix this.

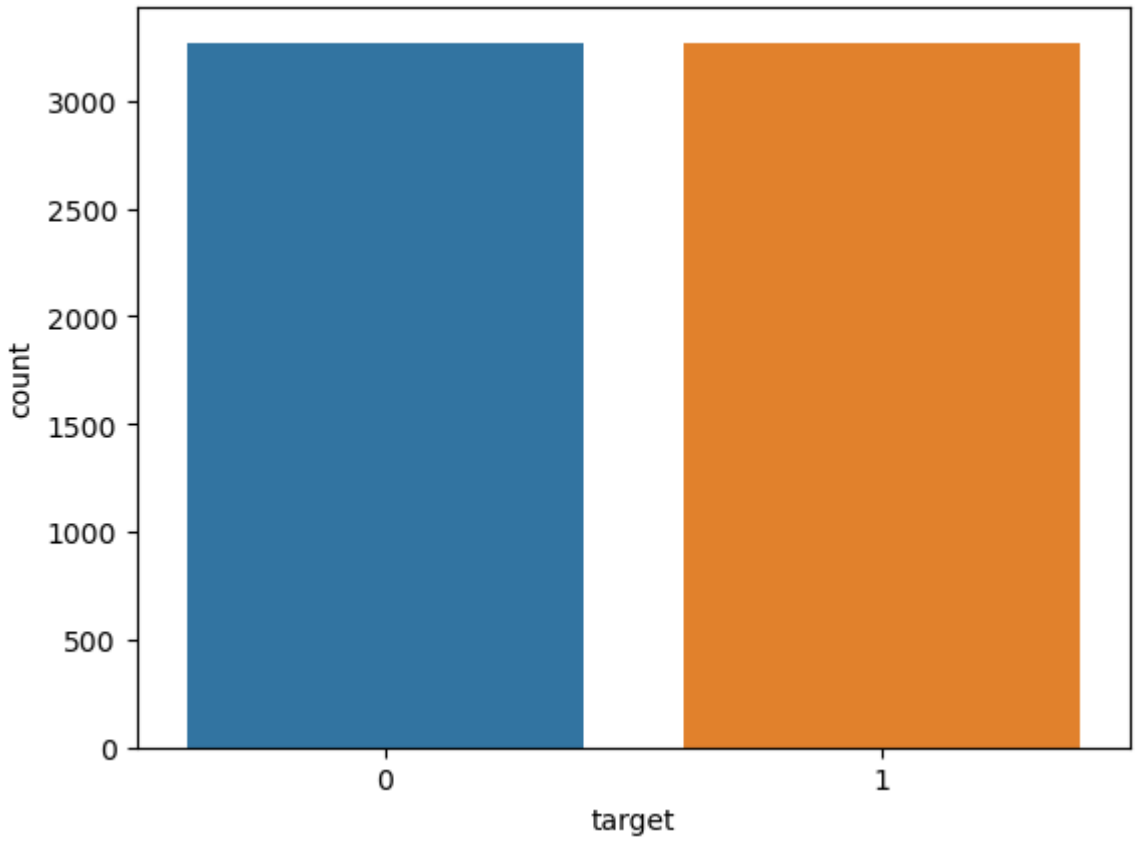
Data Preparation

Use Random Sampling to create an even outcome distribution

In [9]:
```
DisasterTweets = RawDisasterTweetData[RawDisasterTweetData['target'] == 1]
NonDisasterTweets = RawDisasterTweetData[RawDisasterTweetData['target']==0]
NonDisasterTweetsSample = NonDisasterTweets.sample(n = len(DisasterTweets), random_stat
print("Number of Disaster Tweets:", DisasterTweets.shape[0])
print("Number of Non-Disaster Tweets:", NonDisasterTweetsSample.shape[0])
```

```
Number of Disaster Tweets: 3271
Number of Non-Disaster Tweets: 3271
```

In [10]:
```
TrainingTweetsDF = DisasterTweets.append(NonDisasterTweetsSample).reset_index(drop = Tr
sns.countplot(x = 'target', data = TrainingTweetsDF)
plt.show()
```



In [11]:
```
print("Training Tweets shape:", TrainingTweetsDF.shape)
TrainingTweetsDF.head()
```

```
Training Tweets shape: (6542, 5)
```

Out[11]:

	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

Only the text and target features are needed. Remove other fields.

Drop Unnecessary Features

In [12]:
```
TrainingTweets = TrainingTweetsDF[['text','target']]
TrainingTweets.head()
```

Out[12]:

	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

	text	target
3	13,000 people receive #wildfires evacuation or...	1
4	Just got sent this photo from Ruby #Alaska as ...	1

Separate the Text and Target Values

In [13]:
```
X = TrainingTweets['text']
```

OneHotEncode the Target Values

In [14]:
```
TargetTrain = OneHotEncoder(sparse = False).fit_transform(TrainingTweets['target']\
                                                          .to_numpy().reshape(-1,1))
TargetTrain
```

Out[14]:
```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

Use the UniversalSentenceEncoder on the Tweet Text

In [15]:
```
X_train = []
for tweet in X:
    Embedded = UniversalSentenceEncoder(tweet)
    EmbeddedTweets = tf.reshape(Embedded, [-1]).numpy()
    X_train.append(EmbeddedTweets)
X_train = np.array(X_train)
```

In [16]:
```
# Do the Same for the testing data
TestTweets = TweetTestData['text']
X_test = []
for tweet in TestTweets:
    Embedded = UniversalSentenceEncoder(tweet)
    EmbeddedTweets = tf.reshape(Embedded, [-1]).numpy()
    X_test.append(EmbeddedTweets)
X_test = np.array(X_test)
```

In [17]:
```
print("Training Set Shape:", X_train.shape)
print("Testing Set Shape:", X_test.shape)
```

```
Training Set Shape: (6542, 512)
Testing Set Shape: (3263, 512)
```

Model Building

In [18]:
```
model = Sequential()
model.add(Dense(128, input_shape = (X_train.shape[1],), activation = 'relu'))
model.add(Dropout(rate = 0.65))
model.add(Dense(128, activation = 'sigmoid'))
model.add(Dropout(rate = 0.65))
model.add(Dense(64, activation = 'sigmoid'))
model.add(Dropout(rate = 0.65))
model.add(Dense(2, activation = 'sigmoid'))

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               65664
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 130
=================================================================
Total params: 90,562
Trainable params: 90,562
Non-trainable params: 0
_________________________________________________________________
```

Model Training

Hyperparameter Tuning

In [19]:
```
callback = EarlyStopping(monitor = 'val_accuracy', patience = 2)

history = model.fit(X_train, TargetTrain,
                    epochs = 20, batch_size = 32,
                    validation_split = 0.15, callbacks = callback,
                    verbose = 1, shuffle = True)
```

```
Epoch 1/20
174/174 [==============================] - 34s 8ms/step - loss: 0.7647 - accuracy: 0.540
1 - val_loss: 0.8670 - val_accuracy: 0.0265
Epoch 2/20
174/174 [==============================] - 1s 3ms/step - loss: 0.5821 - accuracy: 0.6974
- val_loss: 0.4695 - val_accuracy: 0.8340
Epoch 3/20
174/174 [==============================] - 0s 3ms/step - loss: 0.4845 - accuracy: 0.7831
- val_loss: 0.5761 - val_accuracy: 0.7862
```

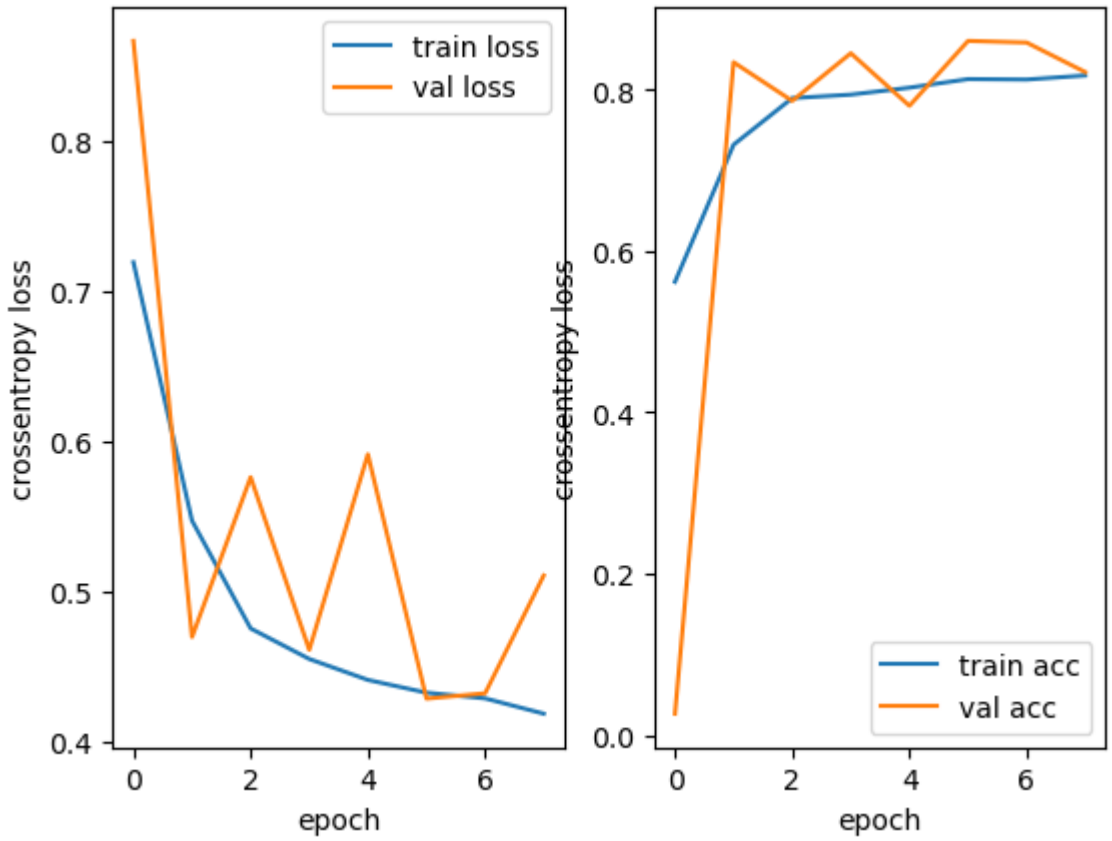
```
Epoch 4/20
174/174 [==============================] - 0s 3ms/step - loss: 0.4509 - accuracy: 0.7963
- val_loss: 0.4610 - val_accuracy: 0.8452
Epoch 5/20
174/174 [==============================] - 1s 3ms/step - loss: 0.4407 - accuracy: 0.8027
- val_loss: 0.5914 - val_accuracy: 0.7800
Epoch 6/20
174/174 [==============================] - 0s 3ms/step - loss: 0.4406 - accuracy: 0.8051
- val_loss: 0.4285 - val_accuracy: 0.8605
Epoch 7/20
174/174 [==============================] - 0s 3ms/step - loss: 0.4134 - accuracy: 0.8260
- val_loss: 0.4320 - val_accuracy: 0.8585
Epoch 8/20
174/174 [==============================] - 0s 3ms/step - loss: 0.4143 - accuracy: 0.8186
- val_loss: 0.5106 - val_accuracy: 0.8218
```

Model Evaulation

In [20]:

```
lossplot = plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label = 'train loss')
plt.plot(history.history['val_loss'], label = 'val loss')
plt.xlabel('epoch')
plt.ylabel('crossentropy loss')
plt.legend()

accplot = plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label = 'train acc')
plt.plot(history.history['val_accuracy'], label = 'val acc')
plt.xlabel('epoch')
plt.ylabel('crossentropy loss')
plt.legend()
plt.show()
```



Save and Load Model

In [21]:
```
model_url = "C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Sentiment A
model.save(model_url)
```

In [22]:
```
NLPModel = load_model(model_url)
```

Predictions

In [23]:
```
predictions = NLPModel.predict(X_test)
predict = pd.DataFrame(predictions,
                       columns = ['Non-Disaster', 'Disaster']).drop('Non-Disaster',
                                                                    axis = 1)
predict.head()
```

Out[23]:

	Disaster
0	0.791299
1	0.950866
2	0.824096
3	0.912066
4	0.987068

In [24]:
```
TweetTestData['target'] = predict['Disaster'].apply(lambda x: 1 if x >= 0.5 else 0)
TweetTestData.head()
```

Out[24]:

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

In [25]:
```
SubmissionDF = TweetTestData[['id','target']]
SubmissionDF.head()
```

Out[25]:

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

In [26]:
```
SubmissionDF.to_csv("C:/Users/tedda/Desktop/Data Science Portfolio/Machine Learning/Sen
```