Module 9 Assignment ML RNN Neural Nets – NLP with Disaster Tweets

Dan Noel

MSDS 422

May 29, 2022

For me the challenge of this week's assignment involved exploring the data and then developing the best way of cleaning the data such that it would be easy to process via a RNN/transformer encoder model.

Having gained much more knowledge of Kaggle, throughout the last ten weeks I was able to find an approach to cleaning and balancing the data that led to decent results. My first few attempts to vectorize tweets were ineffective at breaking through the median because of my inability to reduce the noise in the data.

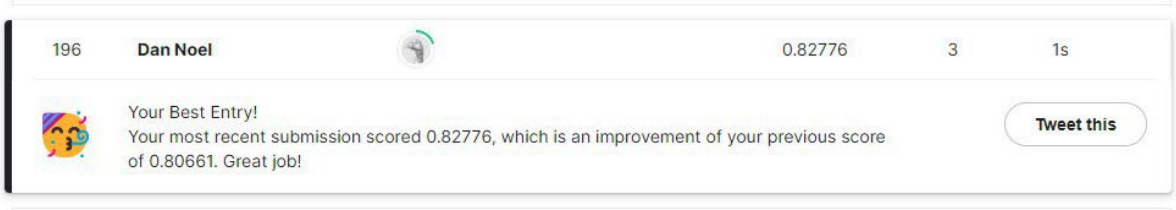
I found the following URL most helpful for explaining some of the various alternative options for building primary tools and getting started with hyperparameters.

https://www.kaggle.com/code/tuckerarrants/disaster-tweets-eda-glove-rnns-bert/notebook#IV.-More-Complex-Model(s)

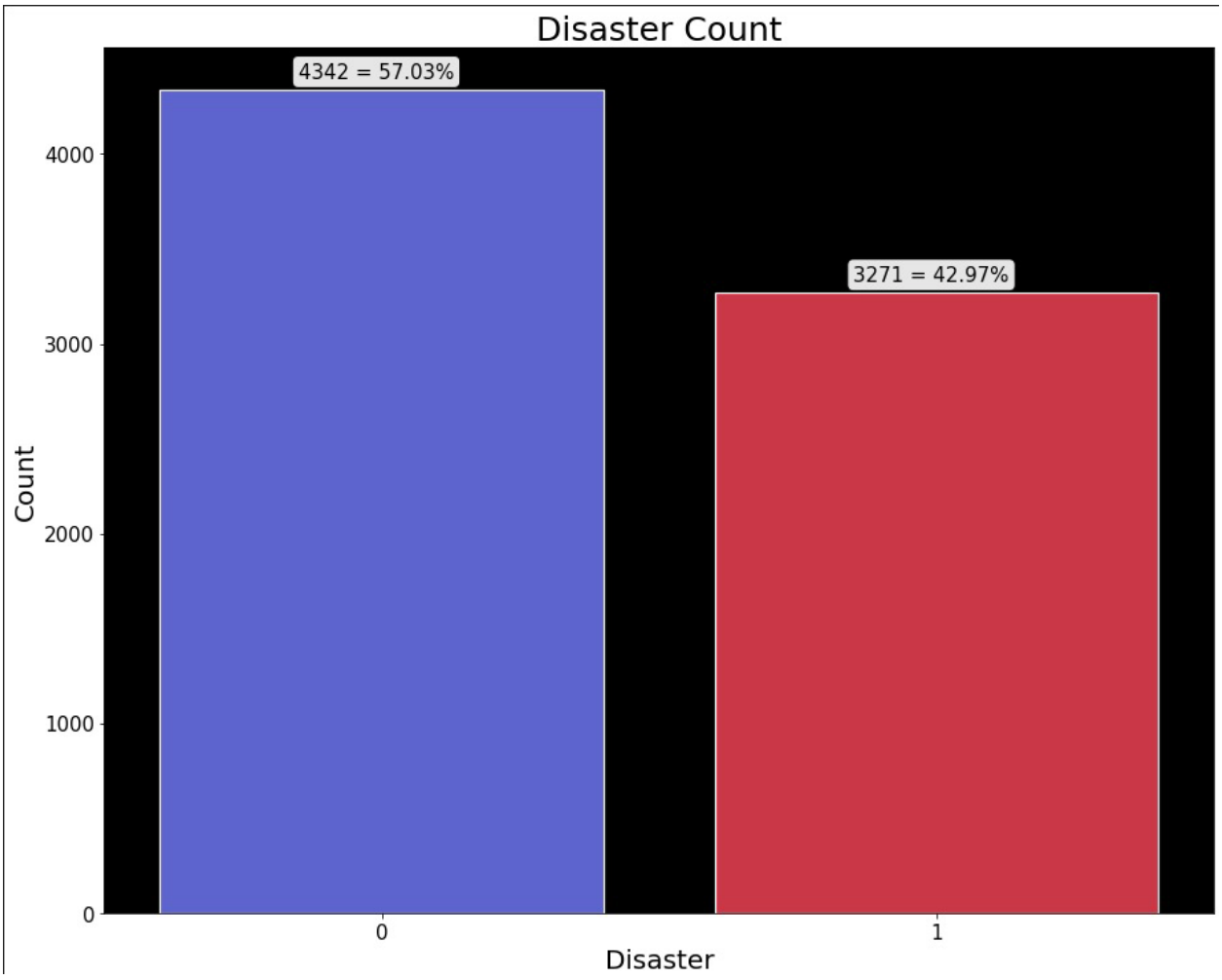
Interestingly, this approach involves a great deal of data cleaning including transforming contractions into whole words and declaring the simple LSTM model.

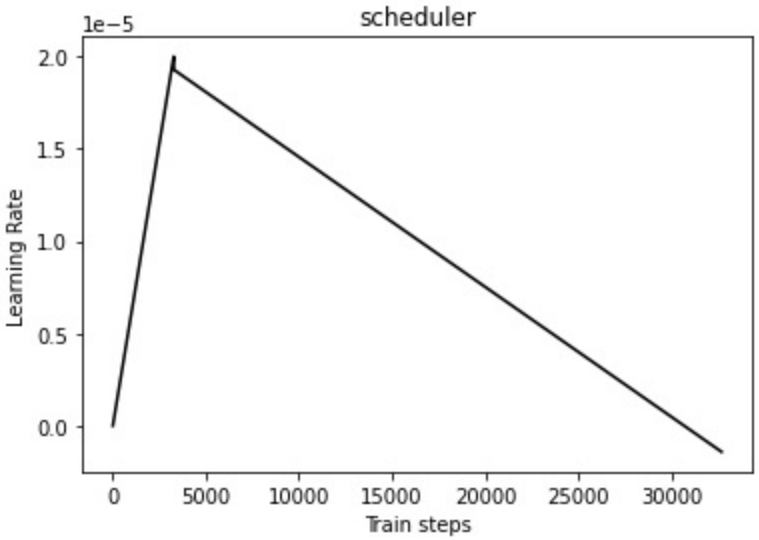
The important part of this assignment involved an approach could clean data (see Appendix 3). And then the actual Classification algorithm (Appendix 4)

**Appendix 1 – Kaggle Metrics**



Appendix 2 – EOD - Metrics




scheduler
1e−5
2.0
1.5
1.0
0.5
0.0
Learning Rate
0
5000
10000
15000
20000
25000
30000
Train steps

## Appendix 3 – Cleanup Code

```
# %20 is the URL encoding of space, let's replace them with '_'
def re_encode_space(input_string):
    return None if pd.isna(input_string) else input_string.replace('%20', '_')


# Let's try to find hastags
import re

def find_hash_tags(input_string):
    hash_tags = re.findall(r"#(\w+)", str(input_string))
    return ','.join(hash_tags)


# Let's turn hashtags to normal words
def re_encode_hashtags(input_string):
    return None if pd.isna(input_string) else input_string.replace('#', '')


# Let's remove URLs from the tweets
def remove_links(input_string):
    res = input_string
    urls = re.findall(r'(https?://[^\s]+)', res)
    for link in urls:
        res = res.strip(link)
    return res


# Let's remove the state abbreviations
def state_renaming(input_string):

    states = {
        'AK': 'Alaska',
        'AL': 'Alabama',
        'AR': 'Arkansas',
        'AZ': 'Arizona',
        'CA': 'California',
        'CO': 'Colorado',
        'CT': 'Connecticut',
        'DC': 'District_of_Columbia',
        'DE': 'Delaware',
        'FL': 'Florida',
        'GA': 'Georgia',
        'HI': 'Hawaii',
        'IA': 'Iowa',
        'ID': 'Idaho',
        'IL': 'Illinois',
        'IN': 'Indiana',
        'KS': 'Kansas',
        'KY': 'Kentucky',
        'LA': 'Louisiana',
        'MA': 'Massachusetts',
        'MD': 'Maryland',
        'ME': 'Maine',
        'MI': 'Michigan',
        'MN': 'Minnesota',
        'MO': 'Missouri',
        'MS': 'Mississippi',
```

```
58          'MT': 'Montana',
59          'NC': 'North_Carolina',
```

```
        'ND': 'North_Dakota',
        'NE': 'Nebraska',
        'NH': 'New_Hampshire',
        'NJ': 'New_Jersey',
        'NM': 'New_Mexico',
        'NV': 'Nevada',
        'NY': 'New_York',
        'OH': 'Ohio',
        'OK': 'Oklahoma',
        'OR': 'Oregon',
        'PA': 'Pennsylvania',
        'RI': 'Rhode_Island',
        'SC': 'South_Carolina',
        'SD': 'South_Dakota',
        'TN': 'Tennessee',
        'TX': 'Texas',
        'UT': 'Utah',
        'VA': 'Virginia',
        'VT': 'Vermont',
        'WA': 'Washington',
        'WI': 'Wisconsin',
        'WV': 'West_Virginia',
        'WY': 'Wyoming'
    }

    result = input_string

    if  isinstance(input_string,  str):
        input_candidates = input_string.split(', ')

        if len(input_candidates) > 1:
            for candidate in input_candidates:
                if candidate in states.keys():
                    result = states[candidate]

    if input_string in states.keys():
        result = states[input_string]

    return result
```

## Appendix 4 – Bert Classification Model

```
from tensorflow.keras.optimizers import Adam

from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score

from sklearn.model_selection import StratifiedKFold

def build_bert_classifier():
    input_ids = tf.keras.layers.Input(shape=(max_tweet_length,), dtype=tf.int32, name="input_ids")
    input_mask = tf.keras.layers.Input(shape=(max_tweet_length,), dtype=tf.int32, name="attention_mask")
    embeddings = bert(input_ids,attention_mask = input_mask)['pooler_output']
    net = tf.keras.layers.Dropout(0.1)(embeddings)
    net = tf.keras.layers.Dense(128, activation='relu', name='pre-clf')(net)
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(1, activation=None, name='classifier')(net)
    return tf.keras.Model(inputs=[input_ids, input_mask], outputs=net)

skf = StratifiedKFold(n_splits=5)
train_average_score = 0
validation_average_score = 0
validation_oof_predictions = np.zeros((len(X['input_ids'].numpy())))

loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
optimizer = Adam(learning_rate=schedule)
epochs = 5

# It's a good practice to predict the test set on every fold
# And average the predictions over the folds
averaged_test_predictions = np.zeros((test_array['input_ids'].shape[0]))

# It's standard practice to use Stratified k-fold cross validation
# so we're also using it here
for fold_n, (train_idx, test_idx) in enumerate(skf.split(X['input_ids'].numpy(), y)):
    X_train_ids = X['input_ids'].numpy()[train_idx]
    X_train_att = X['attention_mask'].numpy()[train_idx]
    y_train = y[train_idx]

    X_test_ids = X['input_ids'].numpy()[test_idx]
    X_test_att = X['attention_mask'].numpy()[test_idx]
    y_test = y[test_idx]

    # Re-build the model at every fold to "reset" it
    model = build_bert_classifier()
    model.layers[2].trainable = True

    model.compile(optimizer=optimizer,
                  loss=loss)

    model.fit(x={'input_ids':X_train_ids,'attention_mask':X_train_att},
              y=y_train, batch_size=32, epochs=epochs)

    train_predictions = model.predict({'input_ids':X_train_ids,'attention_mask':X_train_att})
    validation_predictions = model.predict({'input_ids':X_test_ids,'attention_mask':X_test_att})

    train_score = roc_auc_score(y_train, train_predictions)
```

```
    validation_score = roc_auc_score(y_test, validation_predictions)

    train_average_score += train_score / 5
    validation_average_score += validation_score / 5
validation_oof_predictions[test_idx,] = (validation_predictions > 0.5).astype(int).flatten()

    print(f'Fold: {fold_n}, train auc: {train_score:.3f}, validation auc: {validation_score:.3f}')

    test_predictions = model.predict({'input_ids':test_array['input_ids'],

 'attention_mask':test_array['attention_mask']}).flatten()
    averaged_test_predictions += test_predictions / 5

print(f'Train average: {train_average_score:.3f}, validation average: {validation_average_score:.3f}')
print(f'OOF Accuracy Score: {accuracy_score(y, validation_oof_predictions)}')
```

**Appendix 5 – Tokenization Code**

```
from nltk.tokenize import TweetTokenizer
# The tokenizer is responsible to turn a string of words
# into a list of tokens (words) for which we'll get their
# vector representation (embeddings)
tknzr = TweetTokenizer(
    preserve_case=False,
    reduce_len=True,
    strip_handles=True,
)


def tokenize_tweets(tokenizer, input_text):
    tokens = list(tokenizer.tokenize(input_text))
    tokens = [re.sub('[^A-Za-z0-9]+', '', i) for i in tokens]
    return tokens

original_data['tokens'] = original_data['text']
original_data['tokens'] = original_data['tokens'].apply(lambda x:
tokenize_tweets(tknzr, x))

test_data['tokens'] = test_data['text'].apply(lambda x: tokenize_tweets(tknzr, x))

# We'll pad all embeddings to match the length of the biggest tweet
# in order to account for the variability in tweet length
# Later on the model is going to mask the padded values, so that
# they won't influence the result
max_tweet_length = max(original_data['tokens'].apply(lambda x: len(x)).max(),
                       test_data['tokens'].apply(lambda x: len(x)).max())

X = original_data['text'].tolist()
y = np.asarray(original_data['target'].tolist()).astype(np.float32)

test_array = test_data['text'].tolist()


```