Capstone Project

Machine Learning Engineer Nanodegree

Gary Chen

November 15th, 2021

Definition

Twitter has become an immediate source for news and emergency. With an influx of information every second, important events are announced real time on Twitter before it is broadcasted on news and televisions. One application of this phenomenon is that disaster relief organizations and polices can programmatically monitor Twitter to detect emergencies in order to respond in a timely fashion. Natural Language Processing Techniques (NLP) are able to increase the accuracy of correct classification, especially semantics and Word Sense Disambiguation[1].

In this project, I looked into the Disaster Tweets on Kaggle. The goal of this project is to correctly identify a disaster based on a person's word. For example, a person tweeted "The sunset tonight is ablaze!". This tweet does not indicate a disaster, as the word "ablaze" is used in a metaphorically way. The is clear to a human right away, but it is less clear to a machine.

We first need to transform the text into numbers so that the machine is able to understand. This features engineering will involve 4 steps:

- Text cleaning: Common text cleaning procedures like replacing special characters and Stop Words removal
- Tokenization: Tokenizing all sentences
- Creating vocabulary and Transforming text: We will create a vocabulary of the tokens. Then, the Tokenized sentences will be mapped to unique ids.
- Word embedding: Generate word vectors by using Global Vectors for Word Representation (GloVe)[2]

[1] M. Kanakaraj and R. M. R. Guddeti, "Performance analysis of Ensemble methods on Twitter sentiment analysis using NLP techniques," Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015), 2015, pp. 169-170, doi: 10.1109/ICOSC.2015.7050801.

[2] Global Vectors for Word Representation: https://nlp.stanford.edu/projects/glove/

We will use LSTM (Long short-term memory) to learn whether one text indicates a real disaster (1) else (0). LSTM is preferred over normal Feed Forward Neural Network (FFNN) as it captures the sequence on inputs (in this case, text sequences). The model will be trained on the 75% of the training sample, and the rest of the 25% will be used for validation. The evaluation metric we used to estimate our model performance will be F1 score, which is a harmonic mean of the precision and recall. F1 score is a preferred metric in unbalanced class problem as it addresses both the precision and robustness of our classifier.

Lastly, we will use the final model and perform inference on the testing sample. Then, we will submit the result back to Kaggle for evaluation and scoring.

Analyze

The training sample contains 7613 rows with 5 data columns. The 5 data columns are:

1. Id: A unique identifier for each tweet
2. Text: The text of the tweet
3. Location: The location the tweet was sent from
4. Keyword: A particular keyword from the tweet
5. Target: Binary indicator denotes whether a tweet is about a real disaster (1) or not (0)

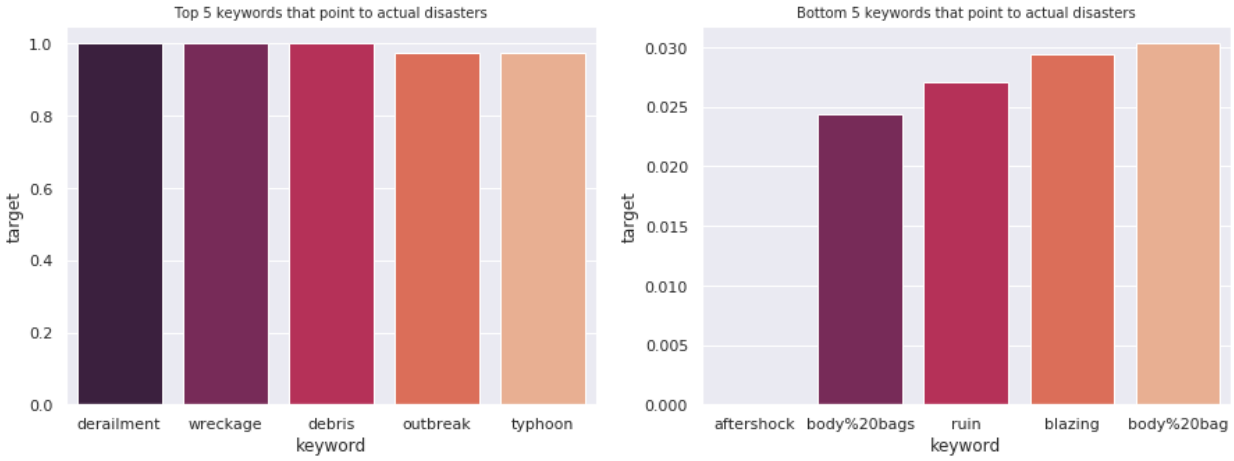
We check the distribution of our dependent variable “Target”. The distribution shows that out of 7613 rows, 3271 or 43% tweets are pointing to an actual disaster, and 4342 or 57% are not.

```
[17]: train_df['target'].value_counts()

[17]: 0    4342
      1    3271
      Name: target, dtype: int64

[18]: train_df['target'].value_counts(normalize = True)

[18]: 0    0.57034
      1    0.42966
      Name: target, dtype: float64
```

We check the fill rate of these data columns, as a column with low fill rate will potentially be troublesome during the model training. The result shows that the “Location” column has only 1 – 33% = 67% fill rate, while the other data columns are close to 100% fill rate. We fill the nulls in the “Location” columns with dummy values.

```
[19]: print(train_df.isna().mean())

id          0.000000
keyword     0.008013
location    0.332720
text        0.000000
target      0.000000
dtype: float64
```

We check the distributions of “Target” by each of the “Keyword” groups. The top 5 keywords that have the highest percentage of actual disasters are derailment / wreckage / debris / outbreak / typhoon. This makes sense as these words are less ambiguous and they normally point to actual disasters. The bottom 5 keywords with the lowest percentage of actual disasters are aftershock / body bags / ruin / blazing. These are all ambiguous words and can mean completely different things when put in different context.



We check the most common words mentioned in the text columns. The result shows that the most common words are mostly “Stop Words” that doesn’t provide any information. We will work on removing these stop words during our text cleaning phase.

```
[35]: # Check the most common words in the text columns
from collections import Counter
results = Counter()
train_df['text'].str.lower().str.split().apply(results.update)
print(results.most_common(10))

[('the', 3207), ('a', 2135), ('in', 1949), ('to', 1934), ('of', 1814), ('and', 1405), ('i', 1336), ('is', 930),
('for', 880), ('on', 834)]
```

Implement

Text feature engineering

We summarize the text feature engineering in the below 4 steps.

1. Text cleaning

In this step, we lower case each text and remove any special characters. For example, “I love that dog #love” becomes “I love that dog love”.

2. Tokenization:

   Tokenizers divide strings into lists of substrings. For example, “I love that dog love” becomes [“I”, “love”, “that”, “dog”, “love”].

3. Vocabulary:

   In this step, we create a dictionary object that records every word that appears in the text. Then, we map each word to a unique ID. After this step, the original text string has converted to an list of integers with the length explicitly specified.

4. Word embedding:

   Work embedding is an algorithm that transform each word into word vectors. The word vector is a learned representation of the original word, and it has several benefits. One of the major benefits is that this representation captures the linguistic or semantic similarity of the two words through Euclidean distance[3].

   We can also perform math operation like addition and subtraction with word vectors. For example, adding “King” and “Woman”, and subtracted the “Man” word vectors would result in “Queen” work vectors.

   We will utilize the glove.6B.50d pre-trained word vectors. This file will have 400K words and each with a vector length of 50. We will make this into an embedding layer in LSTM, so that each word will be embedded when pass through the network.

[3] https://nlp.stanford.edu/projects/glove/

```
[20]: # Check a word embedding
print('King: ',model.get_vector('king'))
result = model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
print('Most similar word to King + Woman: ', result)
```

```
King:  [-0.49346   -0.14768    0.32166    0.056899   0.052572   0.20192
 -0.13506   -0.030793   0.15614   -0.23004   -0.66376   -0.27316
  0.10391    0.57334   -0.032355  -0.32766   -0.2716     0.32919
  0.41305   -0.18085    1.5167     2.1649    -0.10278    0.098019
 -0.018946   0.027292  -0.7948     0.36631   -0.33151    0.2884
  0.10436   -0.19166    0.27326   -0.17519   -0.14986   -0.072333
 -0.54371   -0.29728    0.081491  -0.42673   -0.36406   -0.52035
  0.18455    0.44121   -0.32196    0.39172    0.11952    0.36979
  0.29229   -0.42954    0.46653   -0.067243   0.31216   -0.17216
  0.48874    0.2803    -0.17577   -0.35101    0.020792   0.15974
  0.21927   -0.32499    0.086022   0.38927   -0.65638   -0.67401
 -0.41896    1.2709     0.20857    0.28315    0.58238   -0.14944
  0.3989     0.52681    0.35714   -0.39101   -0.55372   -0.56642
 -0.15762   -0.48004    0.40448    0.057518  -1.0157     0.21755
  0.073296   0.15237   -0.38362   -0.75308   -0.0060254 -0.26232
 -0.54102   -0.34347    0.11113    0.47685   -0.7323     0.77597
  0.015216  -0.66327   -0.21144   -0.42964   -0.7269    -0.067968
  0.50601    0.039817  -0.27584   -0.34794   -0.0474     0.50734
 -0.30777    0.11594   -0.19211    0.3107    -0.60075    0.22044
 -0.36265   -0.59442   -1.2046     0.10619   -0.60278    0.21573
 -0.35362    0.55473    0.58094    0.077259   1.0776    -0.1867
 -1.5168     0.32418    0.83333    0.17366    1.1232     0.10863
  0.55889    0.30799    0.084318  -0.43178   -0.042287  -0.054615
  0.054712  -0.80914   -0.2443    -0.076909   0.55216   -0.71896
  0.83319    0.020735   0.020472  -0.40279   -0.28874    0.23758
  0.12576   -0.15165   -0.6942    -0.25174    0.29591    0.4029
 -1.0618     0.19847   -0.63463   -0.70843    0.067943   0.57366
  0.041122   0.17452    0.19431   -0.28641   -1.1363     0.45116
 -0.066518   0.82615   -0.45452   -0.85652    0.18105   -0.24187
  0.20153    0.72298    0.17415   -0.87328    0.69815    0.024706
  0.26174   -0.0087155 -0.39349    0.13801   -0.39299   -0.23057
 -0.22611   -0.14407    0.010511  -0.47389   -0.15645    0.28601
 -0.21772   -0.49535    0.022209  -0.23575   -0.22469   -0.011578
  0.52867   -0.062309 ]
Most similar word to King + Woman:  [('queen', 0.6978678107261658)]
```

LSTM

RNN is a neural network architecture that can use internal state to process sequence of inputs, which includes time series and text sequences. It can preserve the sequential information between elements in the same sequences. The text data in our dataset involves sequential information of words. Therefore, after words are converted to word vectors, we can use them with RNN.

One of the disadvantages of RNN is that it faces difficulty with long-term dependency in input sequences. LSTM[4], a specific architecture of RNN, could partially solve the vanishing long-term gradients as the LSTM units allow gradients to also flow unchanged.

The LSTM architecture is presented below:

1. Embedding Layer

[4] https://people.idsia.ch//~juergen/rnn.html

This embedding layer will have an input dimension of the length of the tokenize word in our text data (length = 150). The output dimension will be 50 as we are representing each word with a vector length of 50 using the pre-trained weight matrix from GloVe.

2. Two bidirectional LSTM layer

   Two LSTM unit with bidirectional wrapper. A bidirectional LSTM can preserve information from both past and the future, and it has shown very good results as they can understand the context better.

3. Dense layer with Sigmoid function

   Fully connected NN layer with Sigmoid activation function. Since we have a binary classification problem, the Sigmoid activation will output a probability of each text pointing to a real disaster.

The training parameters includes:

1. Training length (number of epochs)
2. Batch size (how many texts to pass through the network during one single training step)
3. Optimizer (which algorithm to use for learning)
4. Number of neurons in each layer

Results

The final training parameters were chosen because they performed the best on the validation dataset. We chose a training epoch of 10 and a batch size of 64. The optimizer we chose is Adam, which is an optimizer that combines the success of both the Momentum and Adagrad algorithm.

We trained the LSTM on 75% of the training sample, and evaluate its performance on the rest of the 25%. The evaluate metric we will be using is F1 score, but we will also look at other common classifier metrics such as Precision, Recall and Accuracy. Below is the definition of these metrics:

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

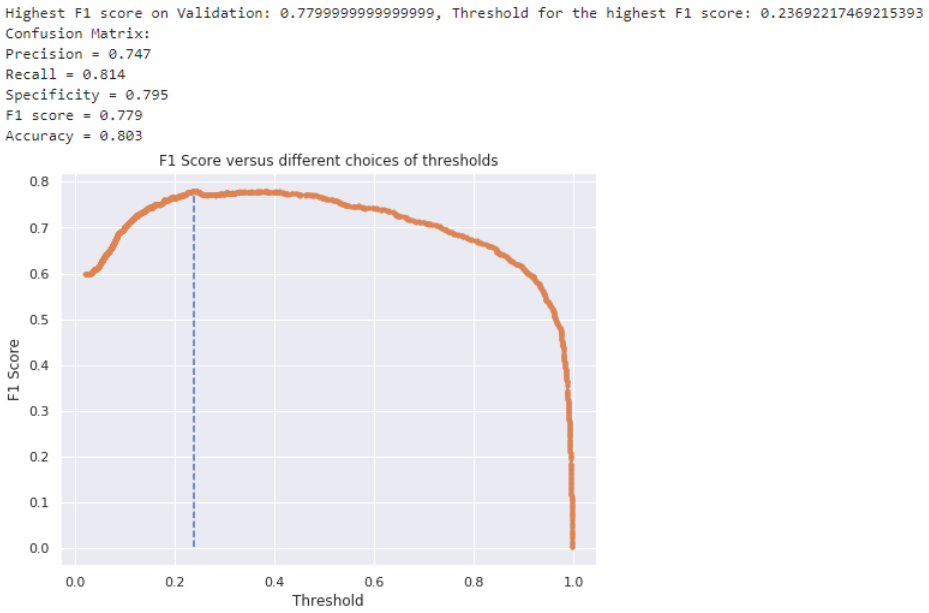
$$\text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

$$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{tn} + \text{fp} + \text{fn}}$$

$$\text{F}_1 \text{ score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The LSTM model outputs a probability of each text pointing to an actual disaster. To calculate what is the best probability threshold, we produce a ROC curve which is a diagnostic plot that evaluates a set of probability predictions made by a model on the validation samples. For each probability, we then calculate the corresponding Precision, Recall and F1 based on the prediction. At last, we chose the optimal probability threshold that returns the highest F1 score possible.

The below graph plots the ROC curve on the 25% validation sample. It shows that the highest F1 score we were able to achieve is 0.78, and the optimal probability threshold is 0.236. The result also shows that the Precision = 0.747, Recall = 0.814, and the Accuracy = 0.803.



Conclusion

To summarize, we perform feature engineering on the text by performing text cleaning, tokenization, and word embedding using GloVe. We then fit the word vectors into a LSTM Keras model which has two layers of bidirection LSTM unit and a dense layer with Sigmoid activation function. We then evaluate the model performance by looking at the F1 score of the model prediction against different probability threshold.

The next steps that could be tested to improve the model performance includes:

1. Additional text features like Term Frequency – Inverse Doc Frequency (TD-IDF) and N-grams
2. Bidirectional LSTM with Attention. The idea of attention aims to search for “a set of positions in a source sentence where the most relevant information is concentrated”.

3. Remove Emojis from the text data as it is quite common for a text to have emojis inside and the text cleaning process currently in place is not handling it