Capstone Project

Mohamed Ashry[*a]

PROJECT OVERVIEW
This project relies on a Kaggle competition âĂIJReal or Not? NLP with Disaster TweetsâĂİ that its goal is to classify the tweets whether it is real or fake disasters. It would be very beneficial to solve this problem as it would help the relief organizations or news agencies to know about these disasters and try to help the people or report the news. And I find it very useful for me as my work might help others and it will enrich my knowledge by learning more about NLP

PROBLEM STATEMENT
The goal is to create a model that able to classify the tweets to whether it is a real\fake disaster. Deploy this model to be able to use it in web application or mobile application.

the tasks involved are:

1)EDA to gain more insights about the data.

2)Visual EDA.

3)Preprocess and clean the data to make it better for the model.

4)build RNN.

5)Train the model and deploy it.

the final product is predicted to be useful as an API for health organizations to classify the tweets.

METRICS
We Would use 3 metrics to measure the model performance.

1)Accuracy.

2)Precision\Recall.

3)ROC\AUC.

DATA EXPLORATION
The dataset is provided by Kaggle.com. It has around 7613 data instances. Which consists of five features;

1)id

2)keyword and it is highlight keywords from the tweet.

3)location and that is from where the tweet was written.

4)Text and it is the tweet that we want to classify.

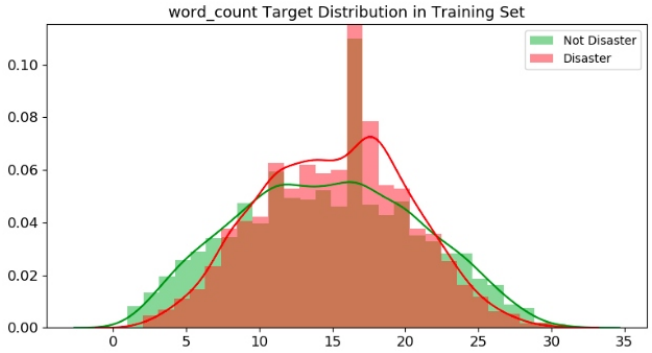
5)target 0 if it is fake 1 otherwise

Each tweet may contain hashtags, mentions, URLs, non-english characters. But the text is written in english.

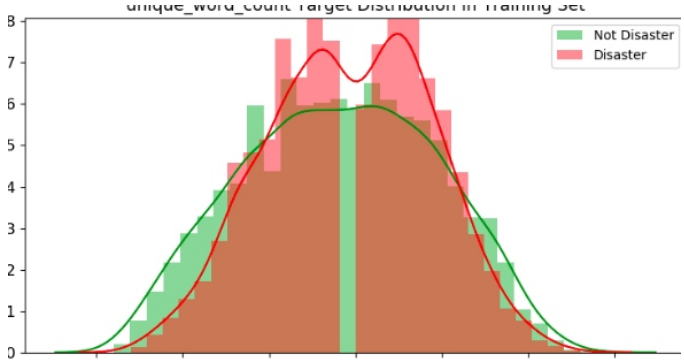
EXPLORATORY VISUALIZATION

These plots shows some distribution of some characteristics of the data.

For example word_count in the disaster and non-disaster tweets.

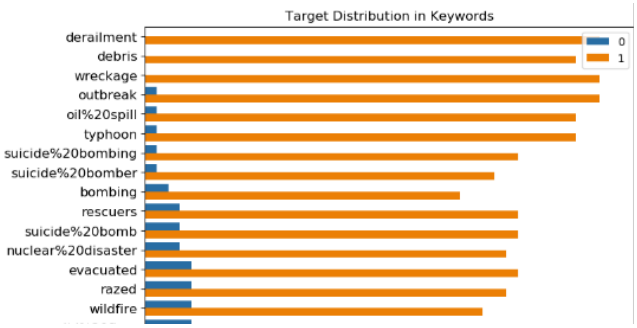


or unique_word_count in the tweets.



And there are a lot more examples in the https://github.com/mohamed-ashry7/Disaster-Tweets-Classifier/blob/master/Twitter-Nlp.ipynb

Also there are the distribution of the words in all tweets.



The full picture is found in the notebook.

ALGORITHMS AND TECHNIQUES
The classifier is Recurrent Neural Network. Which is state-of-the-art algorithm to analyze text data.

The following parameters would be tuned.

1)epochs(Training length)

2)Size of hidden Layers.

3)Batch Size

4)Learning rate

The NN architecture is so simple. It is consisted of the following layers:

Embedding -> LSTM -> Linear

We Would use Sigmoid function as an Activation function.

BENCHMARK
I would use LinearLearner model from SageMaker as a Benchmark to measure how NN is doing well. We would compare between RNN and LinearLearner model using the same matrics which are mentioned above.

DATA PREPROCESSING
1)Relabeling some mislabled tweets

2)The data was divided into train and test datasets

3)Clean the data and that is by:

- Removing special characters
- Expanding Contractions
- Converting typos, slang, informal words to formal ones
- Converting the hashtags to readable ones
- Removing URLs
- Stemming the Words
- Expanding the acronyms

IMPLEMENTATION
Before Forwarding the data to the RNN. We convert the text data into numbers and that is by first clean them and convert them into array of words. And that is by using *tweet_to_words* function. Then counting the frequency of the words and choosing the most `5000` used words in all tweets and store them in a dictionary called *word_dict*. this dict contains two values which represent the Infrequent and no_words words and their values are `1` and `0` respectively.

Then the data is forwarded to the model.

In model directory there is the implementation of the RNN. It contains the following files:

- **model.py** This file contains implementation of RNN. It consists of three layers (Embedding, LSTM, Linear) followed by Sigmoid activation function.
- **train.py** It serves as an entry point when constructing the Pytorch model for training and evaluation.
- **predict.py** It serves as an entry point when constructing Pytorch model for web application deployment.
- **utils.py** It contains two functions (tweet_to_words, convert_and_pad) and they are used when using predict.py.

When training the model, we used BCELoss as the loss function and Adam as the optimizer.

REFINEMENT
For improving the model. I have tried to optimize the values of hyperparamerts.

1)epochs

At first the number of epochs was 10 but the BCELoss values was not converging to a specific value so I increase it up to 40 epochs as BCELoss values were saturating towards a specific value(0.12).

2)size of hidden layer

Since I only used one Fully connected layer so it was better to increase its size to this number as it helped improve its performance. **3)Fixed size for input layer**

I have chosen to make the fixed input size of the rnn to be at maximum 280. Since the max length of a Tweet can not exceed this size. Hence I have lessened the size of the input layer for the network to make it work well on the data.

MODEL EVALUATION AND VALIDATION
the final architecture and hyperparameters were chosen because they performed the best among the tried combinations. So the final model parameters are

- The first embedding layer size is 32
- The hidden layer size is 200
- Learning rate value is 0.001

And to verify the robustness of the model it has been tested with customary examples like "this earthquake is horrible" or "this party is blazing" and it did pretty well on the customary examples also it produced a good accuracy and precision/recall ratio.

JUSTIFICATION
The model did pretty well compared to the benchmark.

By comparing between the RNN and the benchmark with respect to:

- **Accuracy** The RNN has got a higher accuracy which is 75% whereas the benchmark achieved 57%
- **Precision\Recall** RNN achieved 78% for precision and 71% for recall. On the other hand the benchmark achieved 60% and 1.4% for precision and recall respectively.
- **ROC\AUC** As shown in the figures that RNN has 0.75 for AUC and benchmark achieved only 0.5

Figure 1. ROC for RNN.

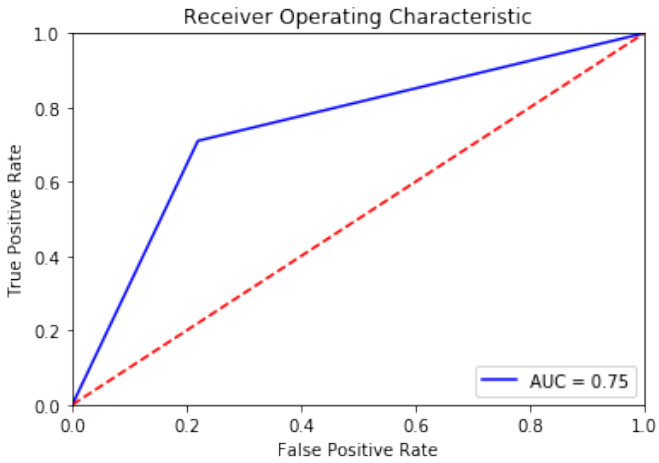


Figure 2. ROC for Benchmark.