Capstone Project

Machine Learning Engineer Nanodegree

Chidera Ozoh

2 April 2021

Predict Disaster Announcement Using Tweets

Definition

Project Overview

In this project, I used tweets data collected from kaggle to determine if a tweet is announcing a disaster or not. As a binary classification problem, the data includes labels of 0 and 1 - 1 means tweet predicts disaster, and 0 means tweet does not predict disaster. Also, as a natural language processing task, the data contains tweets of 7503 training and 3243 testing sets.

Problem Statement

In recent years, social media platforms such as Twitter have been used by users to communicate to the world via tweets. These tweets sometimes contain relevant information which can be used to provide solutions for people. This project addresses the issues of predicting disaster announcements using tweets. To solve this problem, I have collected data containing tweets that have been classified. Hence as a supervised method, a binary classification algorithm will be applied to the data. Since the data contains unwanted strings and words such as urls, tags and stop words, the texts will be pre-processed to remove them. Also, because our model expects a numerical data, bag-of-words (boW) is used to convert token counts to numbers which is then passed to the model. As an NLP

problem, processing the text files into word embeddings is also carried to determine the vocabulary and the words that often occur in the data.

Metrics

The model performed well with an accuracy score of 80 and AUC score of 82. The metrics were measured using sklearn accuracy_score and roc_auc_score. AUC is chosen because of the imbalance in the data, and with the score, it indicates the model can be improved by balancing the outputs in the data.

Analysis

Data Exploration and Visualisation

The data contains training and testing sets of 7613 and 3263 records respectively. With one output and 4 input features, the output contained 4342 labelled 0 and 3271 labelled 1 indicating imbalance in the data. The input features include Id, keyword, location, and text. With the training set containing the target, the testing set does not contain this since it is a Kaggle competition and predicted results will be submitted. Hence to evaluate the model, I split the training set into 6613 train and 1000 validation sets. I analysed the other input features to determine if they are needed for the model. The keyword feature contained 222 unique keywords including 61 NAN records. I noticed there are no empty records in the columns although the location feature has 2533 NAN records.

Exploratory Visualization

The bar plots of the labels indicate an imbalance in the classes with 0 label being more than 1. The keywords were also plotted to see how much each disaster keyword is in the data. I noticed the keywords have not been

stemmed or lemmatized to make similar words to be one. Although the proportion of most of the keywords are in the same range, they can be utilised when for instance an organisation wants to monitor a specific type of disaster. Hence, this feature was not used in building the benchmark model in this project.

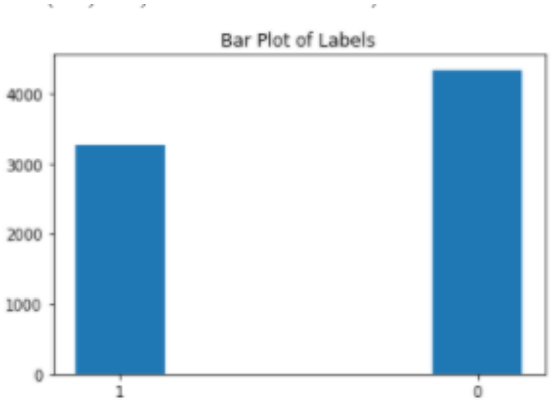

Fig 1.0. Bar plot of targets

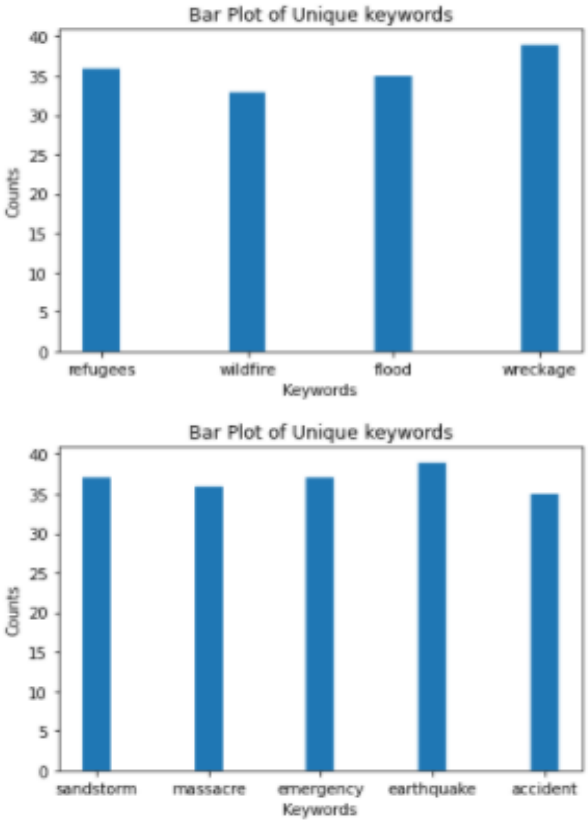

Fig 2.0. Bar Plots of Keywords

Algorithms and Techniques

The NLP techniques and algorithms I used in this project are briefly explained, listed and applied as follows.

- Remove columns not needed for analysis
- Clean and process the texts
- Generate word dictionary
- Convert and pad texts
- Create and upload processed data to s3
- Write model class and apply in training function
- Parse parameters and data to model
- Create a PyTorch estimator
- Fit the input data and train, deploy model
- Evaluate model with Validation data
- Delete endpoint and s3 bucket

Remove columns not needed for analysis: The columns only needed for the training includes text and target. Therefore, other columns are dropped. *Clean and process the texts:* The function texts_to_words was written to process, clean the text and generate tokens of words. Each text in the texts data is cleaned by removing any html tag, unwanted tags, urls, and leaving out only letters. Further, the words are checked to see if they are stopwords and removed. Finally, they are stemmed to check similar words to remove the irregularity of *wreck* and *wreckage* appearing as two words.

Generate word dictionary: I used the count of words in both the train and test data to generate the vocabulary dictionary of the data. This dictionary was later used to create word embeddings used in the model.

Convert and pad texts: In this step, the texts are padded to 100 words and the occurrence of each word of a text in the vocabulary is used to return the length of each sentence or the padding, and count of the texts words in the

vocabulary. This will then be used as the input features for training the model.

Create and upload processed data to s3: The data was first converted to csv and sent to the data directory in the root folder. I then uploaded it to s3 bucket using sagemaker session.

Write model class and apply in training function: The classifier LSTMBinaryClassifier was created and called in the train.py file. The classifier implements three methods - Word embeddings, RNN(LSTM) and Linear Classification. This will be explained more in the methodology section.

Parse parameters and data to model: In the train.py file, I train the model by first parsing arguments such as the parameters of the model, requirements of the model. All these are discussed in the methodology.

Create a PyTorch estimator: With properties such as entry point, role, output path and hyperparameters, I created the estimator for training the model. This is followed by deploying the model and evaluating to determine the performance.

Benchmark

To evaluate the model and determine its performance, the accuracy score and area under curve score (AUC score) are computed. The AUC score is most suitable for this project because it helps measure the imbalance of the data.

Methodology

Data Pre-processing

Briefly explained in the Analysis section, the major preprocessing step taken is generating word vocabulary using Sklearn TfidfVectorizer. Penultimate to this, the words in the texts have been processed using the function in Fig 3 and padded to give the training sets (Fig 4).

```
def texts_to_words(tweet):

    # for each tweet, remove HTML tags if present
    tweet_text = BeautifulSoup(tweet, "html.parser").get_text()

    # convert to texts lowercase and remove urls
    tweet_text = re.sub(r"http\S+", " ", tweet.lower())

    tweet_text = re.sub(r"[^a-zA-Z0-9]", " ", tweet_text.lower())

    words = tweet_text.split()
    # if word is a stopword, remove
    words = [word for word in words if word not in stopwords.words("english")]
    # stem the words
    words = [PorterStemmer().stem(word) for word in words]
    # remove word less than 3 letters
    words = [word for word in words if len(word) > 3]

    # words = " ".join(words)

    return words
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/ec2-user/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# sample
texts_to_words(X_train[700])
```

```
['bekah', 'thank', 'sweat', 'bullet', 'everi', 'time', 'blaze', 'beat']
```

Fig 3.0. Processing words with text_to_words function

```
def convert_and_pad_data(vocabulary_dict, data, pad=100):
    result = []
    lengths = []

    for sentence in data:
        converted, leng = convert_and_pad(vocabulary_dict, sentence, pad)
        result.append(converted)
        lengths.append(leng)

    return np.array(result), np.array(lengths)
```

Fig 4.0. Transform and pad words using vocabulary dictionary

```
train_X[1000]
```

```
array([3128, 1264, 4275,  516, 2564, 4230,    1,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0])
```

```
test_X[500]
```

```
array([1953, 2516,   70,  209,  303,  521, 4933, 2782,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0])
```

Fig 5.0. Results from converting and padding words

Implementation

After processing and uploading the data to s3 bucket, I wrote the LSTMBinaryClassifier module in the model.py file. This module implements the model by using parameters such as the vocabulary size, hidden layers dimension and embedding layers dimension. The vocabulary has a fixed dictionary size of 5000 words found in the data and is passed to PyTorch Embedding module which produces vector representation of words in order to capture their meaning. The output of the Embedding is passed to the long-short term memory (LSTM) module of PyTorch. LSTM network is a recurrent neural network (RNN) that addresses the issue of vanishing gradients in RNN and it is trained using backpropagation through time. Finally, the LSTM results are passed to a Linear layer which produces the final output. One of the complications that occurred is from the shape of input data being passed, this was debugged using cloudwatch.

Refinement

Because the data contains more negative records than positive, when trained the first time it gave an accuracy score of 69% and AUC score of 68%. Also, irregularities due to splitting the training data into train and validation sets contributed to initial poor performance of the model. To improve the performance, I introduced softmax activation on the embedding layer, and dropout of LSTM output before it is passed to the Linear module.

Results

Model Evaluation and Validation

With the pre-processing applied and analysis carried out, the model performed poorly initially. This later increased to accuracy and AUC scores to 75% and 71% approximately when the model and processing steps have been refined and re-examined. Because the test target values were not provided, the test data was not used to evaluate the model.

Conclusion

Reflection and Improvement

This project has tackled the problem of disaster announcement using tweets data. The two major challenges concerning this project is the imbalance of the data labels and the pre-processing of the tweets. There is room for improvement which can be achieved by balancing the targets or increasing the positive target values. Other methods of word processing such as using gensim, and or applying word2vec, BlazingText algorithm can improve the model performance.

References

1. 209 Responses to How to Develop Word Embeddings in Python with Gensim
2. 87 Responses to What Are Word Embeddings for Text?
3. 6.2. Feature extraction — scikit-learn 0.24.1 documentation
4. 160 Responses to How to Make Predictions with Long Short-Term Memory Models in Keras
5. Embedding — PyTorch 1.8.0 documentation
6. LSTM — PyTorch 1.8.1 documentation