

Machine Learning Engineer Nanodegree

Capstone Project - Disaster Tweets

Dimitrios Papadimas

April 10, 2020

Definition

1 Project Overview

The famous social media platform Twitter (<https://www.twitter.com>) has become an important communication channel in times of emergency. The ubiquitousness of smartphones enables people to announce an emergency they're observing in real-time. As a result, tweets could be programmatically monitored in order to identify such events.

However, it is not always clear whether a person's words are actually announcing a disaster or are being used metaphorically. For example, in a tweet such as:

"The sky last night was ablaze!"

the use of the word **"ablaze"** is clearly metaphorical for a *human reader*. On the other hand, a machine lacking human reasoning or context understanding, cannot make this distinction with equal ease.

To address the above challenge, two models were developed, capable of classifying tweets as disastrous (with content that addresses a real catastrophic event/emergency) or normal. For the training of these models, a dataset provided by **Kaggle** was used. (<https://www.kaggle.com/c/nlp-getting-started/data>)

This project is based on a running **Kaggle** competition that can be found in the following url: <https://www.kaggle.com/c/nlp-getting-started>.

2 Problem Statement

The goal of this capstone is the development and deployment of a text classifier that can identify disastrous tweets. Two models were trained and deployed for this task. This process consisted of:

1. Accessing and pre-processing the dataset that contained labeled tweets
2. Downloading pre trained word embeddings in order to transform words/sentences into vectors
3. Train a classifier using a simple Dense Neural Network architecture
4. Train a classifier using a Long Short Term Memory (LSTM) architecture [3]
5. Deploy the model using an html based front-end and a Flask server back-end

The final server will be able to take as input a small piece of text (tweet) and classify it as disastrous or not.

3 Metrics

Since the main goal of this capstone is the development of a binary classifier, the metric that will be used for the validation of the developed models will be accuracy. More precisely, given a set of labeled tweets, accuracy will be the ratio of the correctly classified tweets over the whole set.

Each model will be trained using a subset of the training set. The remaining dataset will be used for calculating the accuracy of the model. In that way, the model is validated on data that it has not processed yet and as a result the model's metric remains unbiased.

Accuracy is a good metric in our case since our dataset can be considered as balanced, as discussed in the Analysis section. In the case of an imbalanced dataset, accuracy would not be a reliable metric since one good argue that simply predicting the dominant class can yield a high accuracy score. However, this is not the case with the dataset used. The *algorithm* of always predicting the largest class in a validation set is not expected, on average, to yield more than 57 % (see Figure 2). Last but not least, the performance of the models on the testing set provided by Kaggle will also be considered.

Analysis

1 Data Exploration

The main dataset, provided by Kaggle, consists of 7613 labeled tweets, together with meta-data. In detail, the dataset consists of the following columns:

- **id**: the tweet's unique id in respect to the dataset
- **location**: the location from where the tweet was uploaded (may be blank)
- **keyword**: a particular keyword from the tweet (may be blank)
- **text**: the textual content of the tweet
- **target**: the label defining whether the tweet refers to a real disaster (1) or not (0)

	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
5	8	NaN	NaN	#RockyFire Update ==> California Hwy. 20 closed...	1
6	10	NaN	NaN	#flood #disaster Heavy rain causes flash flood...	1
7	13	NaN	NaN	I'm on top of the hill and I can see a fire in...	1
8	14	NaN	NaN	There's an emergency evacuation happening now ...	1
9	15	NaN	NaN	I'm afraid that the tornado is coming to our a...	1
10	16	NaN	NaN	Three people died from the heat wave so far	1
11	17	NaN	NaN	Haha South Tampa is getting flooded hah- WAIT ...	1
12	18	NaN	NaN	#raining #flooding #Florida #TampaBay #Tampa 1...	1
13	19	NaN	NaN	#Flood in Bago Myanmar #We arrived Bago	1
14	20	NaN	NaN	Damage to school bus on 80 in multi car crash ...	1
15	23	NaN	NaN	What's up man?	0
16	24	NaN	NaN	I love fruits	0
17	25	NaN	NaN	Summer is lovely	0

Figure 1: dataset example

However, since most of the data from **location** and **keyword** columns are empty (NaN) these columns were dropped from the dataset. Only the textual content of the tweet and the labels were utilized.

(the **id** column was utilized for uploading predictions to Kaggle in order to get accuracy from the competition's testing set)

Another important feature of the dataset that should be mentioned is that it is relatively balanced as far as the labels are concerned. In particular, the tweets labeled as disastrous consist of around 43% of the dataset while normal tweets consist of around 57% (Figure 2). Thus, no actions need to be taken during training for balancing the dataset.

2 Exploratory Visualization

The figure below demonstrates the ratio between labeled tweets inside the dataset. As discussed above, although normal tweets are more than the disastrous ones, this difference is not significant in order to characterize our dataset as unbalanced.

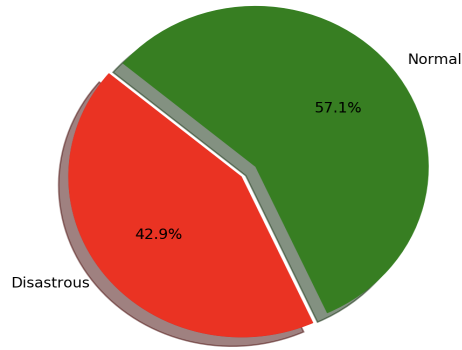


Figure 2: label ratio

Another important characteristic of the dataset that is worth mentioning is the most frequent Out of Vocabulary words (OOV), meaning words that are not included in the model's known vocabulary. The vocabulary that will be used, is described in more detail in the next section, consists of the set of known words from the GloVe embeddings [4, 5], trained in a large corpus from Twitter.

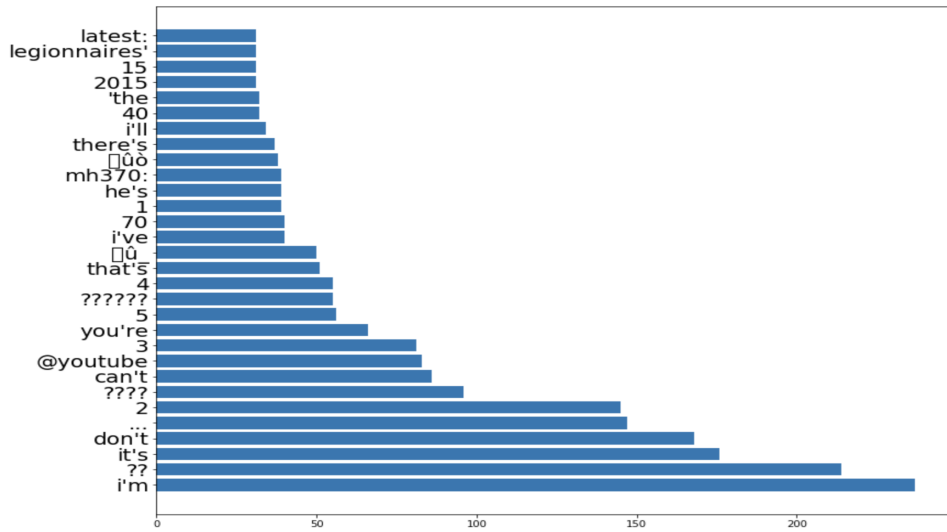


Figure 3: top OOV words

Based on the top OOV words, it can be concluded that:

- Numbers should be considered as OOV words
- Words containing numbers should be considered as OOV words

LSTMs take this procedure a step further by adding a memory feature. Several words can be semantically connected even if they are not directly next to each other in a sentence. For example, in the sentence "On Friday, although I wanted to visit my friends, I stayed home to work on my capstone project.", *Friday* should be associated with *staying at home*, which occurs in the end of the sentence, instead of *visiting friends* which is closer. While traditional RNNs might fail to capture this link, LSTMs implement a hidden memory cell solely for this reason. This memory cell processes the recurrent input from all different words in the sequence and identifies the semantic links several words might have, even if they are not close to each other in the sentence. Thus, an RNN architecture is suitable for the present problem, since its input is text, and an LSTM is a stronger solution compared to a traditional RNN because of its ability to process longer sentences more efficiently by implementing memory.

3.2 Feature Engineering

In the scope of this project, feature engineering consists of transforming the textual content of the tweets into vectors, so that machines can process them. Three algorithms were used, one in a word level and the rest in a tweet level:

- GloVe word embeddings [4] for the representation of each word into a separate word vector
- Universal Sentence Encoder [2] model for encoding the content of a tweet into a 512 dimensional vector
- TF-IDF [6] which is a statistical algorithm for catching relative frequencies of words from a tweet and package them into a bag-of-words vector (where instead of 1/0, each word is represented by its relative frequency)

GloVe GloVe is an abbreviation for Global Vectors. In short, it is a set of words (vocabulary) that are associated with a certain vector representation (called word embeddings). These vectors capture syntactic and semantic relationships between words by taking into account the context around which they appear. A very common use of these vectors is the calculation similarities between words. The term *banana* for instance, is semantically closer to the term *apple*, since they are both fruits and appear in the same context, compared to the term *car*. As a result, when comparing their associated vectors, the distance of *banana* and *apple* will be significantly smaller than that of *banana* and *car*. The way GloVe representations are obtained is by an unsupervised learning approach which uses aggregated word to word co-occurrence statistics from large corpora [5]. After training, the output for each word is a vector that represents its semantic meaning. Training can be an expensive procedure, especially for large corpora of billions of tokens. However, it only occurs once. When the representations are obtained, they can be used to various different NLP problems, without needing to be re-trained. One should simply download the set of vectors, match the word token that needs to be represented with the token from the GloVe vocabulary and extract the associated vector. There are various different word vector representations, such as word2vec. Even GloVe embeddings have different versions based on the corpus that they have been trained on. For this project, GloVe was selected as a word representation algorithm because the version of the word vectors that was chosen is obtained from a huge corpus from Twitter. This is ideal in this context since the final goal of the project is to classify tweets.

The final pre-trained GloVe vectors that were used consisted of 27 billion words with their associated vectors of 200 dimensions each (25, 50, 100 dimensions could also be an option).

Universal Sentence Encoder Similar to the GloVe embeddings, Universal Sentence Encoder(USE) is another way of extracting representations from natural language. The major difference is that instead of extracting word representations, in this approach a pre-trained model is utilized to generate representations for the whole sentence.

The model's input is a string (could be a sentence, a paragraph or even a small document) and the generated output is a vector of 512 dimensions that represents the semantic meaning of the input (for more information, refer to <https://tfhub.dev/google/universal-sentence-encoder/4>).

The USE is a good candidate as a feature extraction algorithm for this projects for a few reasons:

- Although not the state-of-the-art, is it a very powerful model for text classification tasks, as reported on the url above.
- Compared to current state-of-the-art approaches, it is a relatively small model that can be easily deployed
- It is open source and can be easily downloaded from tensorflow hub (<https://tfhub.dev/>)

3.3 Hyperparameter tuning

For optimizing the performance of the models, several parameters can be tuned:

- Training parameters
 - Number of Epochs (training duration)
 - Batch size (number of examples processed before updating the model weights)
 - optimizer (algorithm used to update the model weights)
 - learning rate, weight decay (parameters of the optimizer)
 - learning rate scheduler that dynamically update the learning rate
- Model architecture parameters:
 - Number of fully connected layers
 - Number of stacked LSTM layers (applicable only for the LSTM classifier)
 - Layer parameters (layer input/output size, hidden layer size, dropout)
 - Activation function (ReLu, Sigmoid etc)
- Input parameters
 - Sentence Length to be processed (applicable only for the LSTM classifier)

4 Benchmark

In order to compare the performance of the developed models, a Naive Bayes classifier will be used as a baseline model. The choice of this model was supported by various analysis (i.e. https://sebastianraschka.com/Articles/2014_naive_bayes_1.html) concluding that Naive Bayes classifiers are often used for text classification. Naive Bayes classifiers, which are based on Bayes' theorem of conditional probability, treat each feature as independent from the others, which can be a naive assumption, but is observed to work out well on text data. Thus, in order to create a baseline score, the dataset was splitted into two different sets, a training and a validation set. The

training set was used to train the model and the validation set to calculate the accuracy metric. For the representation of the tweets, both the TF-IDF and the USE algorithm were used. The results are:

Table 1: Benchmark Accuracy

Input Features	Validation Accuracy	Kaggle Accuracy
TF-IDF	68.7 %	69.7 %
USE	72.5 %	73.4 %

Clearly, the USE representation algorithm performs better than the TF-IDF one. This was expected since the representation embeddings are created using an already trained encoder which is able to capture and represent the semantics of sentences (tweets in our case). This is a form of transfer learning [7].

Methodology

1 Data Preprocessing

The data pre-processing is defined in the *src.utils.preprocessing* module, in the code base, and consists of all the steps necessary in order to clean the text before feeding it in the model. This pre-processing steps are defined based on the OOV words described above (see 3), in order to decrease their frequency as much as possible. The pre-processing steps can be summarized as:

1. Lower all characters and remove numeric characters
2. Split text into word tokens
3. Remove noise from corpus caused by twitter related keywords (i.e. x89ûò)
4. Convert all hyperlinks into the "http" token (otherwise they will be labeled OOV)
5. Handle most frequent irregular contractions (i.e. ain't→is not)
6. Expand regular contractions (i.e. don't→do not)
7. Remove non alphabetic characters

Pre-processing should be performed to input text of both models. This includes text that is passed in the model both at training and at inference phase.

It should be noted that removing stopwords is not part of the pre-processing. This is a model design decision, since there are plenty of stopwords that can be valuable to sequence models, such as words like "not", "no" etc. Finally, only for the LSTM model, an additional step is required in order to prepare the input. The text must be transformed into a sequence of indices, each associated with a word token. This is important because during training, these indices will be used in order to fetch the appropriate word embeddings that need to be passed in the model. The sequence of indices must have a fixed length, in order to be passed to the LSTM model. As a result, a sequence length parameter is used. Sentences that contain more word tokens are being pruned to fit this length,

while shorter sentences are padded using 0 as index.

The mapping between indices and tokens is performed from a custom class *Vocabulary* that can be found in the *src.vocabulary.base* module.

2 Implementation

The binary classifier has been implemented in a fit/predict logic. To clarify, under the *src.models* folder, one can find 3 files. The first file, called *tweet.py* implements an abstract classifier called *TweetClassifier* that is initiated by defining the underlying model type together with its configuration parameters. The type of the model can be either **lstm** or **snn**. This class is also responsible for training the underlying model (*fit*) and generate predictions (*predict*). Some other functions of this class include, saving the underlying model's state (*save*), loading a model state from a folder (*load*) as well as calculating the model's accuracy (*get_accuracy*) based on a testing set. The other files implement the two models, *SNNClassifier* and *LSTMClassifier*.

2.1 SNNClassifier

This class can be found under the *src.models.snn* module and implements a Fully Connected Neural Network classifier. The model's architecture is illustrated below:

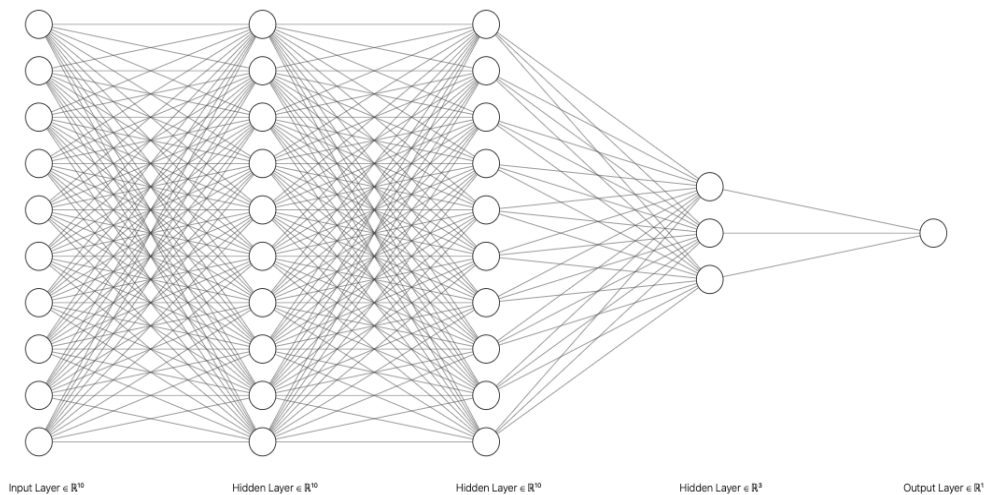


Figure 5: SNNClassifier's architecture

The architecture consists of:

- An input layer of size 512
- Two Fully Connected layers of size 512 where dropout is performed
- One layer of size 32
- The output layer of size 1

2.2 LSTMClassifier

Similarly to the *SNNClassifier*, this classifier's class can be found in the *src.models.lstm* module and implements an LSTM classifier. The model's architecture consists of:

- An input layer of size 200 where index sequences are mapped to embeddings
- An lstm layer with hidden size 200
- A fully connected layer of size 100 where dropout is performed
- The output layer of size 1

2.3 Fit - Training phase

As mentioned above, the model's training is implemented by the *TweetClassifier* object and in particular by the *fit* function. This function expects as input:

- X: numpy array of tweets' textual content
- y: numpy array of the tweets' labels (0 or 1)
- X_val: numpy array of tweets' textual content as a validation set
- y_val: numpy array of the tweets' labels (0 or 1) as a validation set

Based on the type of the model, the X, X_val datasets are initiated and then the appropriate dataloaders are created for the training phase. For the *SNNClassifier* the initiation of the datasets consists of transforming the textual content into vectors using the Universal Sentence Encoder model. On the other hand, for the *LSTMClassifier*, the initiation is the transformation of the text into a sequence of indices, performed by the custom *Vocabulary* class mentioned in the **preprocessing** section.

It should be noted that, in case there is no validation set provided, the training set is split into training and validation set for the training phase. A validation set is always required because during training, for each epoch, the accuracy of the model is evaluated and only the model's state that yields the best accuracy is returned.

For training configuration is:

- Optimizer: The Adam optimizer is being used to update the model's weights
- Criterion: For calculating the predictions' loss, the Binary Cross Entropy Loss is picked
- ReduceLROnPlateau is being used for dynamically tune the learning rate when the model's performance is not improving

Both models were trained for more than 100 epochs and only each model's state that yielded the best results on the validation set was kept.

2.4 Deployment phase

For the deployment of the classifier, a simple Flask [1] server was used as a backend. This server implements an endpoint for predicting the tweets class. Both models are loaded when the server is booted. For the server to work, it requires both models to be already trained and their trained state to be stored in the *model_bin* directory of the repo.

For the front-end, a very basic User Interface was developed using *html*. It implements two input interfaces, one for entering the tweet's text and the other one for choosing the model that will predict the tweet.

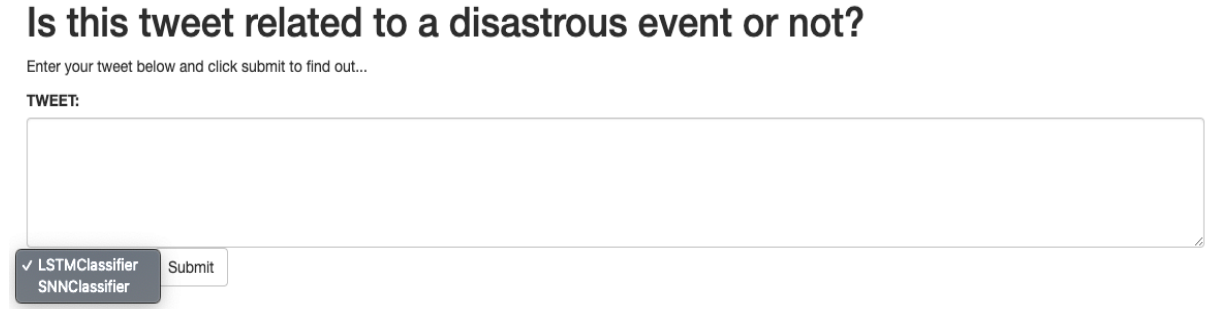


Figure 6: project's UI

3 Refinement

During the model development phase, lots of iteration occurred in order to improve the models' performance. For each iteration, the accuracy metric was calculated using a k-fold cross validation approach with a $K=5$.

From the first iteration, both models reached an accuracy of around 79%, thus outperforming the benchmark set. As a result, in order to further improve the performance of the models, the accuracy of the model in both the training and validation set was compared. In that way, possible overfitting (accuracy on training set was significantly larger than that on the validation set) was identified. Thus, the model was refined in order to decrease the overfitting effect with the following actions:

- Increasing the weight decay in the optimizer
- Increasing the dropout rate in the Fully Connected Layers

Based on these actions, the models' performance was increased by 2-4 % on the validation set, reaching an accuracy of around 81% for the *LSTMClassifier* and 83% on the *SNNClassifier*.

Results

1 Model Evaluation and Validation

For this capstone project, two models were developed, using two different architectures. These architecture are described in detail in the **Methodology** section. Both models were designed,

developed and refined based on feedback obtained by a validation set, using the k-fold cross validation algorithm. The final models however, were further validated for their accuracy, using the testing set provided by Kaggle. In detail, predictions were produced using this dataset as input and the accuracy of these predictions was calculated by the Kaggle’s evaluation platform. The detailed results of the validation of the models, in comparison to the benchmark model are presented below:

Table 2: Model’s Accuracy

Model	Input Features	Validation Accuracy	Kaggle Accuracy
Naive Bayes	TF-IDF	68.7 %	69.7 %
Naive Bayes	USE	72.5 %	73.4 %
SNN	USE	83.3 %	81.3 %
LSTM	GloVe	81 %	79 %

To further verify the models’ accuracy, manual tests were performed using the User Interface and the Flask server. Although these tests were limited, the accuracy obtained was similar to the expected, meaning around 1 tweet was misclassified in every 5 attempts.

2 Justification

As far as Natural Language Processing is concerned, there are lots of different approaches for addressing related problems. As discussed above, for building a text classifier, a Naive Bayes model is considered to be a very simple and effective solution. However, there is intuition behind finally using deep learning architectures for solving this task.

Naive Bayes algorithm utilizes probabilities and statistics in order to classify textual data. In addition, it makes the *naive assumption* that features are independent from each other, which in the scope of a text classifier is never the case. Words have meaning simply when they are put in a sequence.

On the other hand, a deep learning approach is more capable of extracting the semantics out of a sequence of word tokens. LSTMs are successful for this task as they are designed in order to incorporate a token memory throughout the sequence. In addition, the Universal Sentence Encoder used to create the input features for the fully connected classifier is also trained to capture this semantic representation of text into vectors.

When comparing the two produced classifiers, their accuracy is similar. However, one would expect the LSTM model to outperform a swallow Fully Connected Classifier in the task of text processing. In our case, this does not happen due to lack of data. Deep leaning models are *data hungry*, meaning they require lots of training data in order to reach high performance, in terms of accuracy. Thus, the initiation of the input of the Fully Connected Classifier using the USE algorithm acts as a form of transfer learning, adding a lot of power to the classifier.

To clarify, as mentioned in the analysis section, the USE is a pre-trained model that transforms the input sentence into vector representations. These vector representations are then used by our Fully Connected Classifier to classify tweets. The same process of transforming text to vector representations is performed by the LSTM architecture. However, the USE is a powerful model pre-trained using a very big corpus and the only part required to be trained on our dataset is the FC layers. The LSTM on the other hand had to be trained from scratch and since our dataset is limited, could not reach the performance of the USE, as far as encoding a sentence to vectors is concerned.

Conclusion

1 Free-Form Visualization

In the figures below, one can see examples of usage of the tweet classifier. Out of the 5 examples presented, one was misclassified. This reflects the current accuracy of the model; a bit more than 80 %.

Is this tweet related to a disastrous event or not?

Enter your tweet below and click submit to find out...

TWEET:

Giannis was on fire against the #Bulls
#NBA

LSTMClassifier Submit

THIS WAS A NORMAL TWEET

(a)

Is this tweet related to a disastrous event or not?

Enter your tweet below and click submit to find out...

TWEET:

New video captures one man's close encounter with with an intense dust devil last month in Mizoram, India! 🇮🇳 Flag of India "Sound On"

Video sent in by: Lalhneh Zova
#weather #tornado #stormhour #India

SNNClassifier Submit

THIS WAS A NORMAL TWEET

(b)

Figure 7: examples of correctly classified normal tweets from the **LSTM** (a) and the **SNN** (b) classifier

Is this tweet related to a disastrous event or not?

Enter your tweet below and click submit to find out...

TWEET:

TORNADO WARNING SOUTH OF INDIANAPOLIS. TAKE COVER NOW! ANOTHER POSSIBLE TORNADO IS NORTH OF INDIANAPOLIS! #inwx #tornado

LSTMClassifier Submit

THIS WAS A DISASTROUS TWEET

(a)

Is this tweet related to a disastrous event or not?

Enter your tweet below and click submit to find out...

TWEET:

Another nice day and another #wildfire. This time in Saddleworth, @manchesterfire have 4 fire engines, 2 wildfire units plus other appliances. All of which cannot respond to life threatening incidents whilst dealing with this. Please enjoy your 1 hours exercise responsibly

SNNClassifier Submit

THIS WAS A DISASTROUS TWEET

(b)

Figure 8: examples of correctly classified disastrous tweets from the **LSTM** (a) and the **SNN** (b) classifier

Is this tweet related to a disastrous event or not?

Enter your tweet below and click submit to find out...

TWEET:

New video captures one man's close encounter with with an intense dust devil last month in Mizoram, India! 🇮🇳 Flag of India "Sound On"

Video sent in by: Lalhneh Zova
#weather #tornado #stormhour #India

LSTMClassifier Submit

THIS WAS A NORMAL TWEET

Figure 9: example of a misclassified tweet

2 Reflection

The entire project can be described by the following sub-tasks:

1. A challenge was found and analyzed
2. Datasets for this challenge were found and downloaded
3. An exploratory analysis was performed in order to further understand the problem and the data

4. A solution was engineered based on this analysis and a benchmark was created
5. The solution was developed (in our case, the two classifiers were trained)
6. The solution was validated and compared to the benchmark
7. A server was designed and implemented for the solution to be deployed

The reason that NLP tasks are interesting, is the one that makes them also challenging. NLP is highly biased, even for humans. Someones sarcastic comment might be interpreted in various different ways by different humans. In the scope of our problem, the challenge was to capture this biased nature of human communication and allow a machine to interpret it. However, even this interpretation is derived by a biased dataset. The misclassification example illustrated above (see 9) describes one man’s close encounter with a tornado. When discussing about it with various people, there was not a unanimous agreement about whether this tweet should be labeled as disastrous. Thus, a question is raised: *How accurately can a machine perform in tasks that even humans do not agree on, and how accurately can we measure this machine’s performance?*

On the other hand, the power of machine learning can be demonstrated by Figure 7a. The tweet’s content was *”Giannis was on fire against the BullsNBA”*. Both classifiers were able to catch the semantics of the use of the word fire, thus classifying it as normal tweet.

3 Improvement

Although the project reached a relatively high accuracy (more than 80%), I strongly believe that this can be improved. Few suggestions that might improve the solution’s performance are:

- Try different embeddings (BERT could be a good start)
- Try different architectures. Although LSTMs are powerful when it comes to NLP, Transformers are the current state-of-the-art
- Train on more data. The dataset was relatively small for a deep learning task. Thus, I believe that there is room for improvement simply by adding more examples in it

Apart from the model, there is room for improvement to the deployed server as well. The UI only accepts one tweet at the time and expects the user to type it. An upload button could be added in order to allow users to provide a larger amount of tweets and get predictions. In addition, an API could be a very useful addition, as it would allow predictions to occur in an automated way.

References

- [1] <https://flask.palletsprojects.com/en/1.1.x/>.
- [2] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018.
- [3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
- [4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

- [5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- [6] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986.
- [7] Lisa Torrey and Jude Shavlik. Transfer learning.