FINAL PROJECT: CLASSIFYING DISASTER TWEETS

Samuel Hart
samuel.hart@gatech.edu

Tarun Golla
tgolla@gatech.edu

Georgia Institute of Technology

December 10, 2021

ABSTRACT

In the age of social media and smartphones the spread of information on regional, national, and global events has accelerated like never before. Because of this a variety of agencies and organizations are looking to observe social media sites in order to compile actionable intelligence for their objectives. In this paper we will consider one such scenario, in which a disaster relief organization or possibly a news agency is looking to use posts from the microblogging site Twitter to collect information on disasters (ie car accidents, wildfires, earthquakes, and more). After collecting some tweets it would be important for a relief organization or news agency to know whether the tweet is actually talking about a disaster or not. We will propose a modified convolutional neural network for classifying such a collection of tweets and compare it to both simpler and more complex models including pretrained ones such as Google Research's pretrained BERT model for sequence classification.

1 INTRODUCTION

The social media site Twitter has become a staple for disseminating information in the form of mini blogs about global and regional events including ongoing disasters. Due to the fact that everyone in this age owns a smartphone, can create an account on Twitter, and send out tweets as frequently as they want, the volume of text data and information available for a given ongoing event is enormous. For this reason, crisis-response organizations have the opportunity to capitalize on the treasure trove of tweets that come through during a disaster. However, this data can be quite noisy precisely due to the fact that anyone with an internet connection and account, which might as well be everyone today, can send out a tweet. So, to sort through the noise and find the tweets that actually pertain to disasters, Machine Learning and Natural Language Processing (NLP) could possibly stand up to the task and greatly accelerate the filtering process.

As such, there has been much research in the area of using NLP in tweet classification for crisis response. Much of the research centers around using supervised learning methods such as Support Vector Machining (SVM) [4, 6], Convolutional Neural Networks (CNN) [2, 4, 5, 6], K-Nearest Neighbors (KNN) [6], Bidirectional Long Short-Term Memory (Bi-LSTM) [5], and more. In addition to a variance in models, there has also been research in the effect of word embeddings on the performance of these models. Much of the written works makes uses of pre-trained word embeddings such as GLoVe, Word2Vec, and FastText. Overall, much of the research concludes that the use of pre-trained embeddings as a starting point and further task-specific training on these embeddings provides better model performance.

In this project we propose a CNN architecture that is similar to [2] with some minor changes to the structure. We also make use of pre-trained **word2vec** embeddings but add separate pretrained embeddings known as **emoji2vec** [8]. For comparison's sake, we also implemented a Naive Bayes Classifier that takes in the data as a bag of words model as well as using Google's pretrained Bidirectional Encoder Representations from Transformers (BERT) [7].

2 OUR MODEL

For our model we implement a CNN architecture with one convolutional layer with max pooling followed by a hidden linear layer and a logarithmic softmax layer that returns a vector of the probability distribution over the two classes: probability of being a disaster tweet and probability of not being one. The convolution layer makes use of 100 filters with three window heights of 3, 4, and 5. Loss is computed using the Cross Entropy Loss formula which is why to save a step in loss computation we use log-softmax for the last layer.

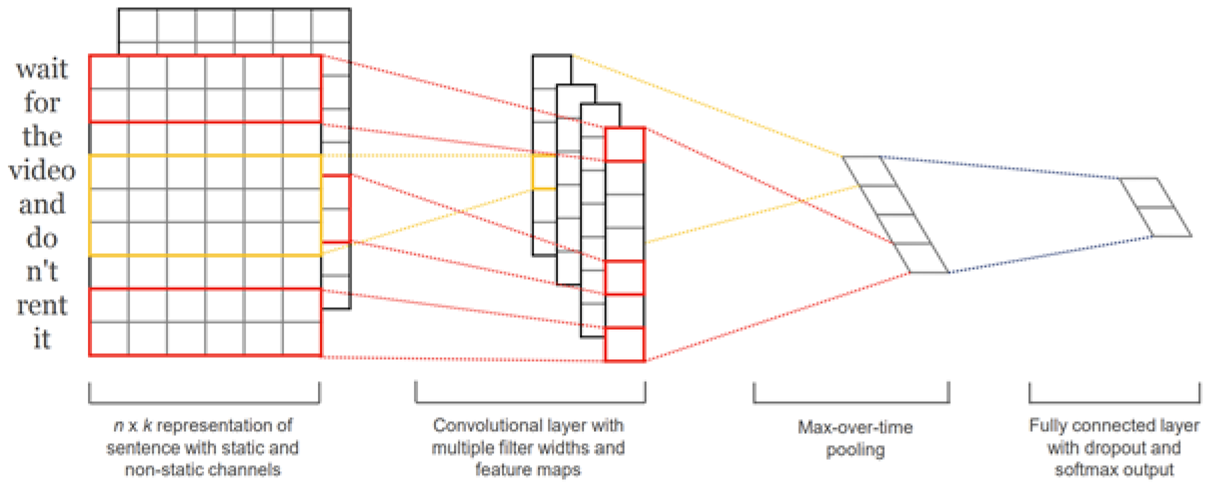
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: The above diagram is taken from [2] and although different from ours, it still provides some idea of our model. Note that for our model there is only channel in the convolution layer emanating from the pretrained embeddings. We also chose to use log-softmax instead of softmax.

The convolution layer works as follows. Note that when we refer to a 'word' in a tweet it may not necessarily be a correctly spelled word and could also refer to an image url, emoji, special symbol, etc. The tweet is taken in and split into words using a Tweet Tokenizer from the Natural Language ToolKit. Once split into words, each word is turned into a word vector using the pre-trained embeddings that will be discussed later. A tweet with $n \leq 128$ words (padded with whitespace to length 128 if necessary) then becomes a matrix with the ith row being the word vector corresponding to the ith word in the tweet. The convolution layer then applies a filter **w** that is applied to a window of 3, 4, or 5 words and outputs a real number known as a *feature*. The filter is applied to each possible window of words to produce a vector of features known as a *feature map*. The pooling operation then takes the maximum feature from each of the filters. The idea behind this is to only keep the most important feature from each feature map. Our convolution layer again makes use of 100 of these filters of varying window sizes 3, 4, and 5 to obtain many maximum features. These are then passed onto the hidden layer and log-softmax layer.

Note that the pre-trained embeddings are not kept static and are adjusted through backpropagation as the model trains. To reduce overfitting we apply a dropout on the units before the log-softmax layer to help prevent dependencies between the hidden units that may come up during the training phase [3]. Essentially, dropout takes in a probability between 0 and 1 and then uses this probability to decide if a given hidden unit should be set to zero. This then allows forward and backpropagation to only affect those hidden units unaffected by the dropout.

3 THE DATA AND EXPERIMENTAL SETUP

3.1 Data, Preprocessing, and Splits

We test our model on the data available on the Kaggle project site [1] which this project idea is based off of. The data consists of 7,613 tweets along with their respective IDs, location, associated keywords, and labels. We started by dropping the keyword and location columns seeing no way they could fit into our model and after finding very little ideas of how to fit them in on the Kaggle leaderboards. All words in the tweet were then converted to lowercase only so as to normalize the words in the tweets. Finally, we split the data into training, dev, and tests sets according to a 70/15/15 split. Furthermore, in addition to measuring accuracy as the proportion of tweets correctly labeled we also make use of F1 accuracy which is given below.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall},$$

where

$$precision = \frac{TP}{TP + FP}, \; recall = \frac{TP}{TP + FN},$$

with TP representing a prediction of 1 matching the true label, FP representing a prediction of 1 and true label of 0, and FN representing a prediction of 0 and true label of 1.

3.2 Pre-trained Word Embeddings

Similar to many other models we decided to initialize our word vectors from pre-trained embeddings that are publicly available and known as the **word2vec** vectors that were trained on 100 billion words from Google News. Due to memory constraints we had to use a slimmed down version of this dataset only containing the most common 10% of the original set. However, our tweets also contained emojis and on the off chance that these emoji could contain valuable information that is relevant to our model we also decided to use a pre-trained word embedding model known as **emoji2vec**. This model transforms an emoji into a 300-dimensional vector by taking each word in the text description of the emoji and adding together their corresponding word vectors from **word2vec** [8]. By incorporating **word2vec** into the making of these emoji vectors the emoji vectors will be close to the word vectors that have similar meaning to the emojis. We also passed on providing embedding vectors for punctuation other than periods, commas, semi-colons, colons, question marks, and exclamation marks. URL tokens, tag tokens using @, and trend tokens using are sent to their own respective randomly initialized vectors. All other out-of-vocabulary tokens are sent to a randomly initialized vector mapped to a '<unk>' token.

3.3 Comparison Models

For performance comparison we also implemented a Naive Bayes Classifier and Google's pre-trained BERT transformer. The Naive Bayes Classifier is implemented using the same TweetTokenizer to split the tweets into tokens and enter them into a vocabulary dictionary mapping indices to unique words. We used a bag of words model to create an occurrence matrix where each row corresponded to a tweets and the entries in the rows were the number of times that the unique word appeared in the tweet, where the column index is the index of the unique word in the dictionary. The Naive Bayes Classifier then computes the numerator of the likelihood of a given tweet being a disaster tweet and the likelihood of it not being a disaster tweet, and classifies it based on which probability is higher. The BERT transformer comes with its own tokenizer that is used to tokenize the tweets. It also uses the Cross Entropy Loss formula. More information on the inner workings of BERT can be found in [7].

4 RESULTS AND FUTURE IDEAS

4.1 Results and Comparisons

Model performances are given below.

Model	Accuracy	F1 Accuracy
Naive Bayes	77.4%	68.1%
CNN	79.2%	76.5%
BERT	81.3%	78.8%

We weren't expecting to beat Google's pre-trained model BERT, however, we were happy with how close we got to its results. It is also worth noting that BERT's transformer model has longer training and testing times compared to our CNN model. An epoch for BERT takes an average of 8x longer than our modified CNN model which could be relevant in a time sensitive situation like classifying disaster tweets.

4.2 Ideas for improvement

Below we give some thoughts for how our model could be improved:

- Firstly, we were unable to make use of Google's full **word2vec** model due to memory constraints. This would allow us to make use of about 10x the number of embeddings we use now. We found that around 12-15% of our word tokens were marked as out of vocabulary and thus limiting their use to the model as they are all mapped to the same random vector.
- We believe there is some degree of overfitting that contributes to our lower accuracy and that this could be minimized by adding more dropout points in the CNN architecture similar to what can be found in [3].
- There is evidence that training our own **word2vec** model on the data using the **gensim** library could provide improvements by creating embeddings better fit for the data [6].

5 CONCLUSIONS

Similar to other research mentioned in this report, Deep Neural Networks could be a useful tool in collecting information regarding disasters on Twitter. Furthermore, we also found that using pre-trained embeddings and allowing further adjustment of these embeddings during training improves performance. However, there is still plenty of research to be done on how to improve these embeddings for task-specific cases.

References

[1] https://www.kaggle.com/c/nlp-getting-started

[2] Y. Kim, "Convolutional neural networks for sentence classification," 2014. [online]. Available: arXiv preprint arXiv:1408.5882. [Accessed Dec. 4, 2021].

[3] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012. [online]. Available: arXiv preprint arXiv:1207.0580. [Accessed Dec. 8, 2021]

[4] D. T. Nguyen, K. A. A. Mannai, S. Joty, H. Sajjad, M. Imran, and P. Mitra, "Rapid Classification of Crisis-Related Data on Social Networks using Convolutional Neural Networks," 2016. [online]. Available: arXiv preprint arXiv:1608.03902. [Accessed Dec. 8, 2021]

[5] R. AlRashdi and S. O'Keefe, "Deep Learning and Word Embeddings for Tweet Classification for Crisis

Response," 2019. [online]. Available: arXiv preprint arXiv:1903.11024. [Accessed Dec. 8, 2021]

[6] H. Li, X. Li, D. Caragea, and C. Caragea, "Comparison of word embeddings and sentence encodings as generalized representations for crisis tweet classification tasks," In Proceedings of ISCRAM Asia Pacific, ISCRAM, 2018.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2019. [online]. Available: arXiv preprint arXiv:1810.04805. [Accessed Dec. 4, 2021].

[8] B. Eisner, T. Rocktäschel, I. Augenstein, M. Bosnjak, and S. Riedel, "emoji2vec: Learning Emoji Representations from their Description," 2016. [online]. Available: arXiv preprint arXiv:1609.08359. [Accessed Dec. 4, 2021].