CS 269 - Final Project Report

Team Members:

- Yu-Chen Lin 705315195 ericlin8545@cs.ucla.edu
- Jo-Chi Chuang 005350427 ashleyashley19951213@gmail.com

Project Title: Real or Not? NLP with Disaster Tweets

Introduction:

Twitter is one of the most important social networks that can even be an important communication channel in times of emergency. With smartphones, people can record the emergency they're experiencing in real-time. Thus, monitoring emergencies with Twitter becomes an interesting topic. In this project, we would like to monitor disasters with tweets and try to make a real-time monitoring disaster system that can be used to help people acquire the information of disaster and take action to protect their lives.

In the task, we will have several training data that consist of information such as keyword, location, and tweet text. For example,

keyword	location	tweet text
aftershock	Oshawa, Canada	#WisdomWed BONUS - 5 Minute Daily Habits that could really improve your life. How many do you already do? #lifehacks http://t.co/TBm9FQb8cW

With the information, we need to identify whether this twitter can reflect a disaster or not. In the task, we will need to achieve higher accuracy with the testing data of the above information.

Related Work

Document/Text classification using Scikit-learn [link]

Since text documents are actually a series of text words, and in order to let machines run algorithms on the data, we need to extract features from text sentences and count the number of times each word occurs in the document. At the end of the preprocessing steps, each unique word will be mapped into a numerical feature vector.

In order to convert a collection of text words into a matrix token counts, Scikit-learn supports feature_extraction.text.CountVectorizer function. By doing fit_and_transform on the data using this component, we are learning a vocabulary dictionary and will get a Document-Term matrix.

To count the frequency of each word in the document, we can use TF-IDF (term frequency-inverse document frequency) statistic method. In Scikit-learn, it also provides *TfidTransformer* function to use.

There are several machine learning models that can be used for text classification. Naive Bayes model is an easy-to-use model for most text classification problems. Also, since the presence of particular words in a document or a post might be directly linked to the results, linear models are also widely used for text classification.

Methodology

Since this project is a typical binary classification problem, we proposed the methods of training Neural Network models to achieve the classification. Thus, we used the popular neural network framework *Tensorflow* to do the models with the toolkit Keras. For the project, we separate the method into several steps, which are Data pre-processing, Model setting & training, Model testing & output file.

- ***Data pre-processing***
  This section includes reading files, removing stop words, tokenization, and padding. Since Keras has provided all the services, we used their built-in function to do the data pre-processing.
- ***Model setting & training***
  In this section, we need to set the model and choose the parameters. In the project, we proposed to compare different embedding methods, including *Keras built-in*, *GloVe*, and the *Universal Sentence Encoder.*
    - *Keras built-in embedding*: Keras built-in embedding is a trainable embedding since it will keep updating the content during the training process.
    - *GloVe:* GloVe is a powerful word vector learning technique that doesn't only rely on local statistics such as local context information of words but incorporates global statistics like word co-occurrence.
    - *Universial Sentence Encoder*: Universal Sentence Encoder is the product of Google, it will transfer the whole sentence into an embedding vector, rather than a vector for a word. The Universial Sentence Encoder provides efficient and accurate results and can be applied to diverse transfer tasks.

  For other settings, please refer to the following table:

Embedding Dimension	100
Dropout Rate	0.3
Kernel Regularizer	L2
Activity Regularizer	L1
Optimizer	Adam
Epoch	20
Batch Size	100

For the training, we use two different methods, which are fully-connected neural network and recurrent neural network (RNN). For the fully-connected neural network, it's just the model that connects each layer. And for RNN, we used Long Short Term Memory (LSTM).

Besides, we also include cross-validation during the training process, and we set the validation split as 0.2. From the process of cross-validation, we can observe the accuracy without submitting our result to the Kaggle website.

- ***Model testing & output file***
  In this section, we just transfer the testing data into the pre-processed format and test them with our trained model. Then output the results as .csv files.

With the procedure, we could train several neural network models and test their accuracy with testing data. To check the accuracy, we just submit our result to Kaggle. In this way, we can compare the performance between different models.

Experiments & Results

We implement deep neural networks using Keras, and compare the results with the three different word embedding methods as mentioned above. First, we implement three identical models except for their word embedding methods so that we can fairly compare how different embeddings affect the performance. Excluding the embedding layer, each model has three layers, with first using relu activation and l1 & l2 regularizers, second with a dropout layer, and last the output layer with sigmoid activation function. All the parameters are identical among all these three models with different embeddings. The performance difference between them can be shown below. Overall, universal sentence encoder outperforms the other two, and GloVe has better performance than Keras built-in embedding. Therefore, we can conclude that using pre-trained word embedding does give a model better initial weights for the training process, and it has a direct influence on the performance.

Moreover, we found that there are several parameters that might be helpful to improve the models, such as pooling methods and the trainable parameter. For the

pooling method, we tried max pooling and average pooling to do the comparison. In the max pooling method, it selects one maximum vector in terms of its norm among the sequence of vectors and uses it as the initial weights. And as for the average pooling method, it would get the average of all the vectors as the initial weights. From the table below, we notice that the sometimes max-pooling does better than average pooling, and sometimes average pooling outperforms. However, the performance difference between these two pooling methods is quite small; thus, we think the pooling method does not affect the performance much.

The second parameter we have is *trainable*. *Trainable* is a boolean type parameter. When it is set to true, the model is allowed to update weights during the training process. When it is false, the embedding layer is excluded from training. Thus, technically, the performance will be better when the *trainable* parameter is set to true. And the results in the table below also show that the accuracy of fully-connected NN models with *trainable* set to true is higher, vice versa. However, surprisingly, we found that, for RNN, the models where *trainable* is set to false will have lower accuracy. We haven't come up with a reasonable explanation for the result, however, it is just our empirical observations.

	Pooling		Trainable		Training		Accuracy
	Max	Average	True	False	Fully-Connected NN	RNN	
Keras built-in word embedding	V		V		V		0.78629
		V	V		V		0.77402
	V		V			V	0.73926
		V	V			V	0.73926
GloVe	V			V	V		0.75664
	V			V		V	0.80265
	V		V		V		0.80368
	V		V			V	0.74948
		V		V	V		0.76073
		V		V		V	0.77505
		V	V		V		0.79856
		V	V			V	0.77811
Universal	Universal			V	V		**0.81288**

Reference

- GloVe: Global Vectors for Word Representation, Jeffrey Pennington and et al., 2018. https://nlp.stanford.edu/projects/glove/
- Universal Sentence Encoder, Daniel Cer and et al., 2018, https://arxiv.org/abs/1803.11175
- Adam: A Method for Stochastic Optimization, Diederik P. Kingma and et al., 2014, https://arxiv.org/abs/1412.6980