Natural Language Processing with Disaster Tweets

0716008 呂沛儒　0716203黃子潔　0716080劉文心

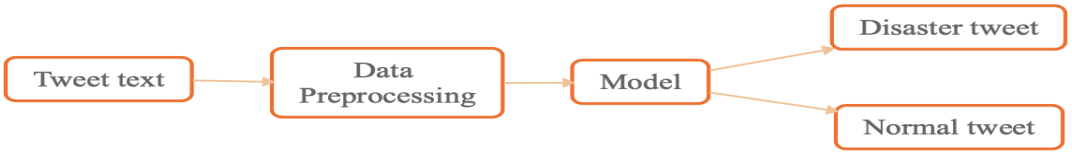
Spring 2021 Introduction to Artificial Intelligence
Final Project Report
Github repo link: link

1 Introduction

During disasters and emergencies, microblogs, social media, have been used by people whether from the private or public sector, local or international community, as a medium to broadcast their messages. Social media are being considered as a means for emergency communications because of their growing ubiquity, communications rapidity, and cross-platform accessibility. The interactions on social media being highly distributed, decentralised and occurring in real time,

Twitter is a social networking service that allows its subscribers to broadcast short messages called tweets. These tweets are used to share relevant information and report news. It is important to solve this problem because tweets provide either first-person observations or bring relevant knowledge from external sources in emergency situations.

However, abundant tweets are promising as a data source, it is challenging to automatically identify relevant messages since tweets are short and unstructured, resulting in unsatisfactory classification. In this project, we tried to find the best model and preprocessing method to classify disaster-related tweets. The performance evaluation is based on the score of Kaggle's submission.



Concept figure of our solutions

2 Related work

A Comparative Analysis of Machine Learning Techniques for Disaster-Related Tweet Classification, published by IEEE Region 10 Humanitarian Technology Conference in 2019, tried to solve the problem we want to solve. The methods they adopted were applying 7 machine learning classifiers and 5 different deep learning models and comparing them. The machine learning classifiers they chose were SVM, Random Forest, Logistic Regression, KNN, Naive Bayes, Gradient Boosting, Decision Tree. The deep learning algorithms they applied were CNN, LSTM, GRU, Bi-directional GRU, and GRU-CNN.

Even though our methods for solving this problem also use machine learning classifiers and deep learning methods, We think we outperform the related work in data preprocessing. In the related work, they applied one preprocessing method for all algorithms. In our methods, we tried more than one way to preprocess data.

3 Methodology

We have tried various methods to solve this problem and the detailed descriptions are down below.

3.1 Machine learning

3.1.1 Preprocessing

In order to transform words to what machine learning models can read, we try to vectorize the tweets by using vectorizer: tools that transform texts to numeric vectors. The vectorizers we chose are Term Frequency-Inverse Document Frequency (TFIDF) and Count vectorizer, which are the two most popular vectorizers in scikit-learn modules.

3.1.2 Model

We tried three machine learning models: RidgeClassifier, LogisticRegression, Gradient Boost algorithm.

RidgeClassifier:

1. Convert target variable into +1 or -1 based on the class in which it belongs to.
2. Build a Ridge() model to predict target variables. The loss function is MSE + L2 penalty Ridge model formula:||y - Xw||^2 + alpha * ||w||^2, alpha=1
3. If the Ridge() regression's prediction value > 0, then predict as positive class else negative class.

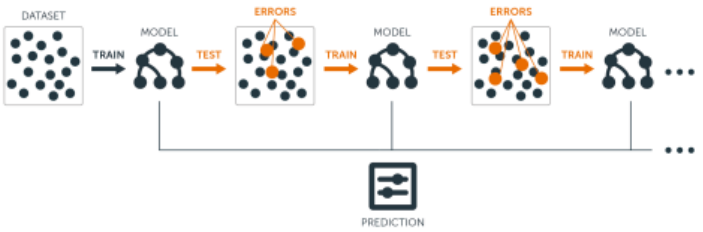
LogisticRegression:

1. Computes the coefficients and intercepts to minimize half of the sum of squares of the coefficients + C times the binary cross-entropy loss, where C is the regularization parameter.
   formula:
   $$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Gradient Boost:

Convert target variable into +1 or -1 based on the class in which it belongs to.

1. Compute pseodo-residuals
2. Fit a base learner to pseudo-residuals
3. Compute multiplier by solving one-dimensional optimization problem
4. Update the model
5. Continue the steps above until iteration ends



We split the original train data to train and test with portion 7:3 to evaluate the model. The field we selected is “text”, which is the content of the tweets. After tuning the model to the ideal condition, we input true test data to the model and judge its performance.

3.2 Bert + Neural Network

3.2.1 Preprocessing

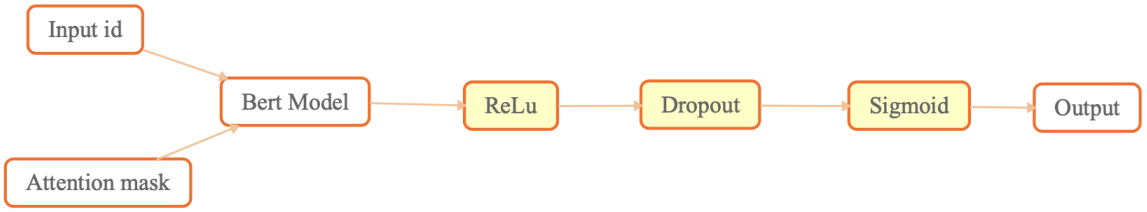
Since not all the words in the tweets are needed for judging whether it's a disaster tweet, we clean the data as the followings:

1. Remove hashtag
2. Remove link
3. Remove non-ascii chars
4. Remove punctuation

3.2.2 Model - Bert + Neural Network

BERT, Bidirectional Encoder Representations from Transformers, is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.Besides, BERT was trained with the masked language modeling (MLM) and next sentence prediction (NSP) objectives. It is efficient at predicting masked tokens and at NLU in general, but is not optimal for text generation.

The following figure is the designation of our method:



The input id and the attention mask is the input of Bert model. They came from pre-trained BertTokenizer.Input id is a 2D array that represents each word in tweets in integer while attention mask is 0 or 1.

As for the setting of the Bert Model, we set the optimizer "Adam", the model as TFBertModel, learning rate 1e-5, and the lost function as binary cross entropy. And most of the settings of the Bert model are the default setting of pre-trained bert-base-uncasedmodel.The Bert model will output an array and send it to the 1st hidden layer of the neural network.

The activation function at 1st layer is ReLu. The second hidden layer is the Dropout layer, it'll randomly set a certain portion of data to 0 while training. Finally, the last layer of the neural network is a dense layer with activation function Sigmoid. It'll output a list of float numbers between 0 and 1. We set the threshold 0.5. If the value > 0.5, it's a disaster tweet; Otherwise, it's not.

We split the original train data to train and test with portion 4:1 to evaluate the model. The field we selected is "text", which is the content of the tweets. After tuning the model to the ideal condition, we input true test data to the model and judge its performance.

3.3 Text Cleaning + ELECTRA

3.3.1 Preprocessing - Text Cleaning

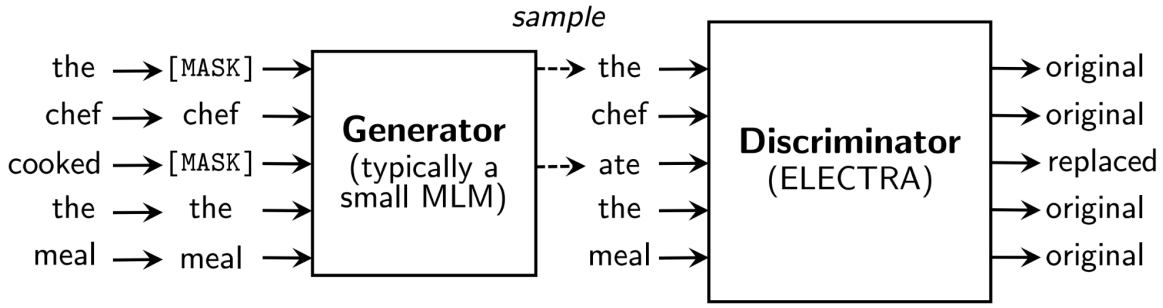
Tweets require lots of cleaning because they may contain words that don't have embeddings. However, it is inefficient to clean every single tweet because that would consume too much time. A general approach must be implemented for cleaning. I make some rules to apply on all the tweets.

1. Punctuations #, @, +, &, -, $, =, <, >, |, {, }, ^, ', (,),[,], *, %, ..., ', :, ; are removed from words(except . ! ?)
2. Urls are removed
3. Replace &, &lt, &gt with &,<,>
4. Remove mentions
5. Remove non ascii chars
6. Contractions are expanded
7. Remove STOPWORDS e.g. am, the, is...

3.3.2 Model - ELECTRA Pre-Training Approach

In Masked Language Modelling(MLM), a certain percentage of the tokens of an input sequence is masked, and the model is tasked with predicting the original token for the masked tokens.

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) is a new pre-training approach which aims to match or exceed the downstream performance of an MLM pre-trained model while using significantly less compute resources for the pre-training stage. The pre-training task in ELECTRA is based on detecting replaced tokens in the input sequence. This setup requires two Transformer models, a generator and a discriminator.



Setup for ELECTRA pre-training

The generator model is trained to predict the original tokens for masked-out tokens while the discriminator model is trained to predict which tokens have been replaced given a corrupted sequence.

The discriminator model is used for downstream tasks and the generator is thrown away after pre-training.

In our method, we use the pre-trained ELECTRA model and tokenizer from "google/electra-base-discriminator" and further fine-tune it on the particular task which is to predict tweets about disasters. We use "AdamW" as the optimizer and use "get_linear_schedule_with_warmup" as the scheduler.

Before sending data into our model, we split the original train data into training dataset(80%) and validation dataset(20%). Then, use the training dataset to fine-tune the model and use the validation dataset to observe whether overfitting is happening. FInally, use the test dataset to judge the performance of the model.

3.4 LSTM

3.4.1 Preprocessing

We do data cleaning to remove outliers and standardize the data so that it can be easily used to create a model.

1. Remove punctuation such as! , ", #, $, %, &, (), *, + ,- , ., /, :, ;, <, =, >, ?, @, [, \,], ^, _, `, {, |, }, ~
2. Remove emojis by using emoticon_fix
3. Fix contractions in English by using contractions.fix e.g. you're -> you are
4. Split each word by using word_tokenize in nltk library
5. Remove stopwords

We choose "text" field and "keyword" to do the preprocessing and do the following steps to transform word to vectorizer that the model can read:

1. Create a dictionary and change the text to integer sequence
2. Add zero padding to make all inputs have the same length.

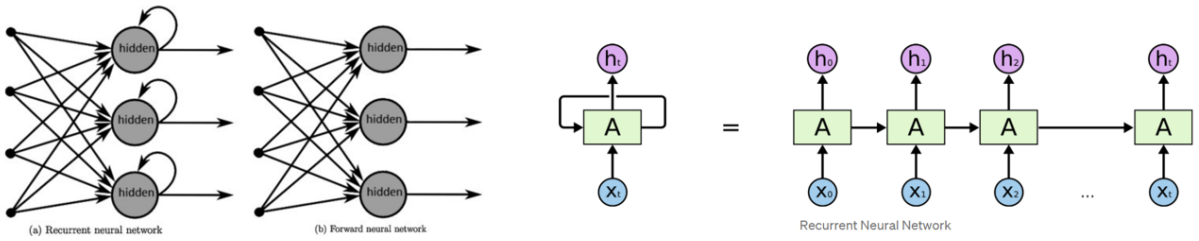
In the training dataset, "target" field contains 0 or 1 which denotes whether a tweet is about a real disaster (1) or not (0). Therefore, we do one-hot encoding on the "target" field so that each value will be a 2-D vector.

3.4.2 Embedding Layer

In order to map semantic meaning into a geometric space and capture precisely the semantic relationship between two concepts, we use word embedding. Each integer will be changed into an n-dimensional vector.

3.4.3 Model - Neural Network

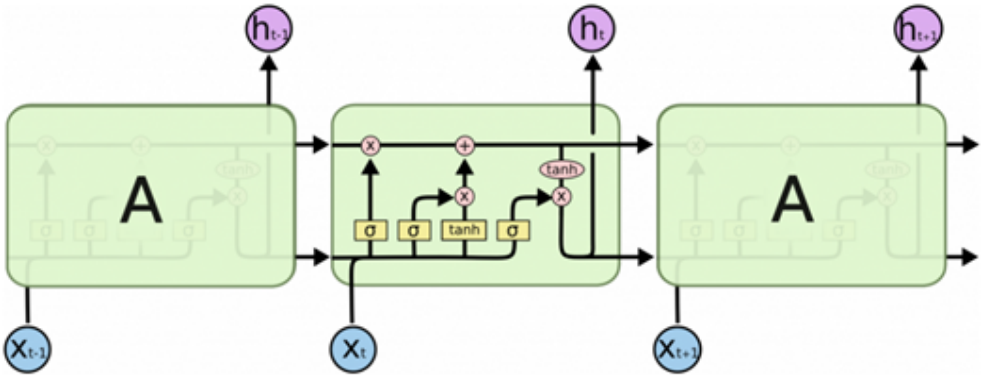
In the Recurrent Neural Network, it can remember sequences. The RNN is different from the FNN. The FNN takes decisions based on only the current input. The RNN takes decisions based on current and previous inputs



This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of a neural network to use for such data.

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task. However, RNNs can't capture the relationship between long gaps of two relevant information. Thus, we use LSTM which doesn't have this problem.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. All recurrent neural networks have the form of a chain of repeating modules of neural network. LSTMs also have this chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

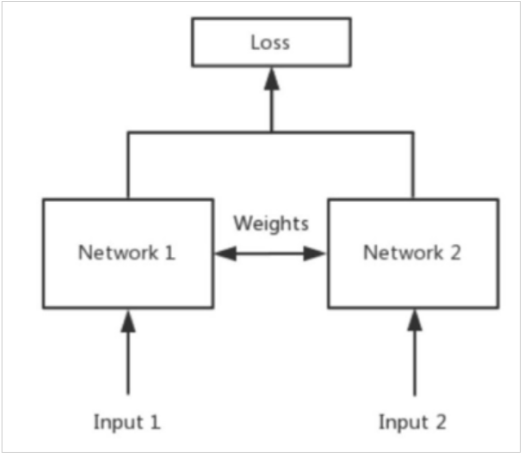


In general, one LSTM cell contains three gates including the forget gate, input gate and output gate. These three gates are used to protect and control the cell state. Based on this structure, LSTM can store the memory state for a long time ago and use them when needed.

In our method, we choose the “text” field as input and split train data into training dataset and validation dataset to evaluate the model. After tuning the model to the ideal condition, we input true test data to the model and judge its performance.

3.4.3 Model - Siamese Network

In our dataset, we also have the “keyword” field but we have only one input in the previous model. Thus, we try a Siamese network which can have two inputs and one output.



In our method, we choose the “text” field and “keyword” field as input and split train data into training dataset and validation dataset to evaluate the model. Network1 and Network2 are both LSTM. After tuning the model to the ideal condition, we input true test data to the model and judge its performance.

4 Experiments

4.1 Machine learning

4.1.1 Result

Algorithm	Vectorizer	Accuracy	Precision	Recall	F1 Score
Ridge Classifier	TFIDF	**0.80049**	0.80112	**0.71255**	**0.75425**
	Count	0.78057	0.78254	0.67760	0.72630
Logistic Regression	TFIDF	0.79436	0.80433	0.68902	0.74222
	Count	0.79988	0.81444	0.69187	0.74817
Gradient Boost	TFIDF	0.78363	0.80526	0.65478	0.72227
	Count	0.79314	**0.82601**	0.65692	0.73182

As the chart shown above, the best machine learning algorithm is Ridge classifier with TFIDF vectorizer. It has 80% accuracy. Besides, each algorithm has its suitable vectorizer. Logistic regression and gradient boost perform better with Count vectorizer while ridge classifier performs better in TFIDF.

4.2 Bert + Neural Network

4.2.1 Hyperparameters

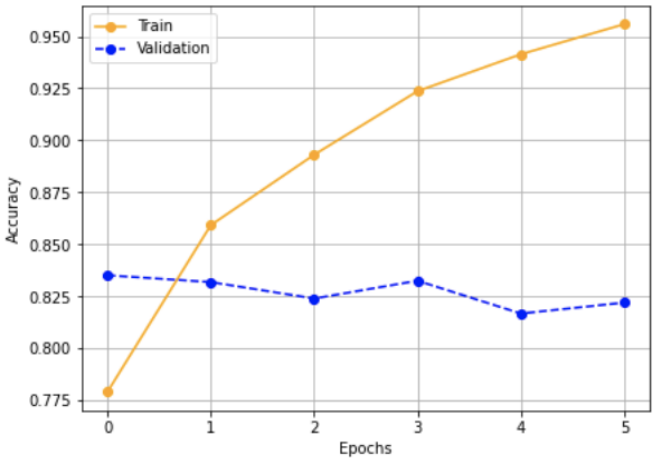
1. Batch Size
   I've tried batch size 4,8,16,32. The one with the best performance is 16.
2. Epoch
   The performance is the best when epoch equals to 2.
3. Dropout rate
   I've tried neural network dropout rate 0.1, 0.2, 0.3. 0.2 is the one with the best performance

4.2.2 Result

1. Batch Size (epoch =2, dropout rate=0.2)

Batch Size	Accuracy	Precision	Recall	F1 Score
4	0.82439	**0.92688**	0.64194	0.75853
8	0.81643	0.78989	0.78031	0.78507
16	**0.83543**	0.86746	0.72825	0.79178
32	0.82439	0.79293	**0.80029**	**0.79659**

2. Epoch (batch size = 16, drop rate = 0.2)
   The graph below is the accuracy of train and validation data when epoch = 6 . It shows that we overfit the data; therefore, we tried epoch from 1 to 3 to see which one best fit the model:



Epoch	Accuracy	Precision	Recall	F1 Score
1	0.80785	0.76559	**0.79672**	0.78085
2	**0.83543**	**0.86746**	0.72825	**0.79178**
3	0.82777	0.84539	0.73324	0.78533
6	0.81520	0.82192	0.72532	0.77185

3. Drop rate (batch size = 16, epoch= 2)

Drop rate	Accuracy	Precision	Recall	F1 Score
0.1	0.82960	0.83203	**0.75606**	**0.79223**
0.2	**0.83543**	0.86746	0.72825	0.79178
0.3	0.82807	**0.87713**	0.69757	0.77712

4.3 Text Cleaning + ELECTRA

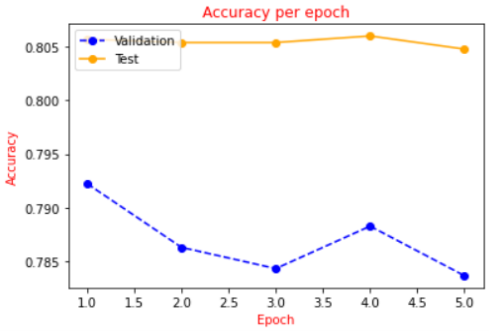
4.3.1 Hyperparameters

1. Batch Size: 4, 8, 16
2. Learning Rate: 2e-6, 2e-5, 2e-4
3. Epoch: 1~5

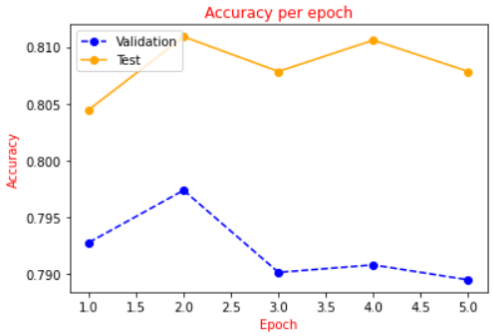
4.3.2 Results

The graph below is the accuracy of validation and test data per epoch with different learning rate and batch size.

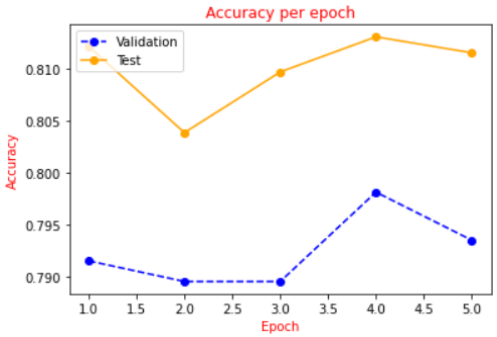
1. batch = 4, lr = 2e-6



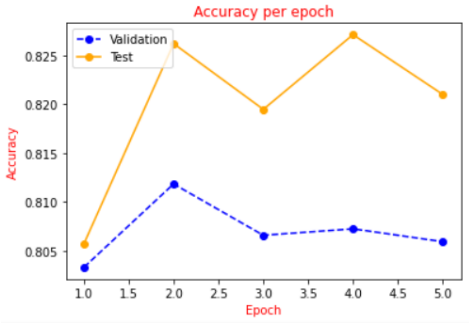
2. batch = 8, lr = 2e-6



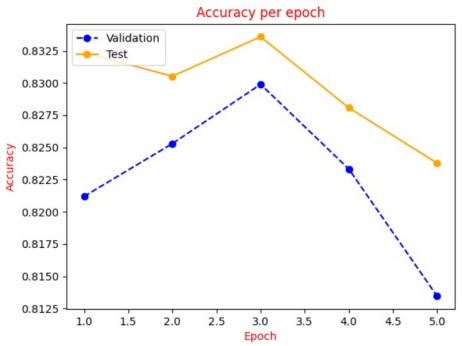
3. batch = 4, lr = 2e-5



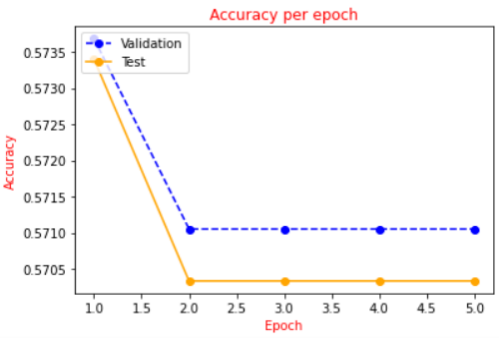
4. batch = 8, lr = 2e-5



5. batch = 16, lr = 2e-5 (Best)



6. batch = 8, lr = 2e-4



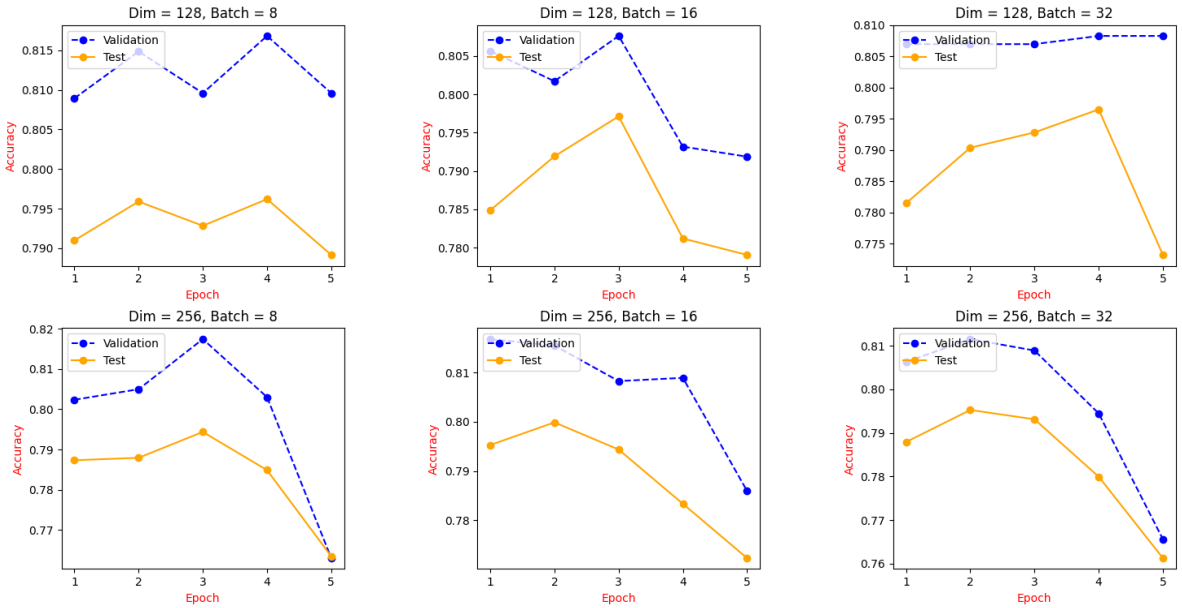
4.4 LSTM

4.4.1 Hyperparameters

1. Batch Size: Tried batch size 8,16,32
2. NUM_EMBEDDING_DIM: Tried 128, 256
3. Epoch: 1 ~ 5

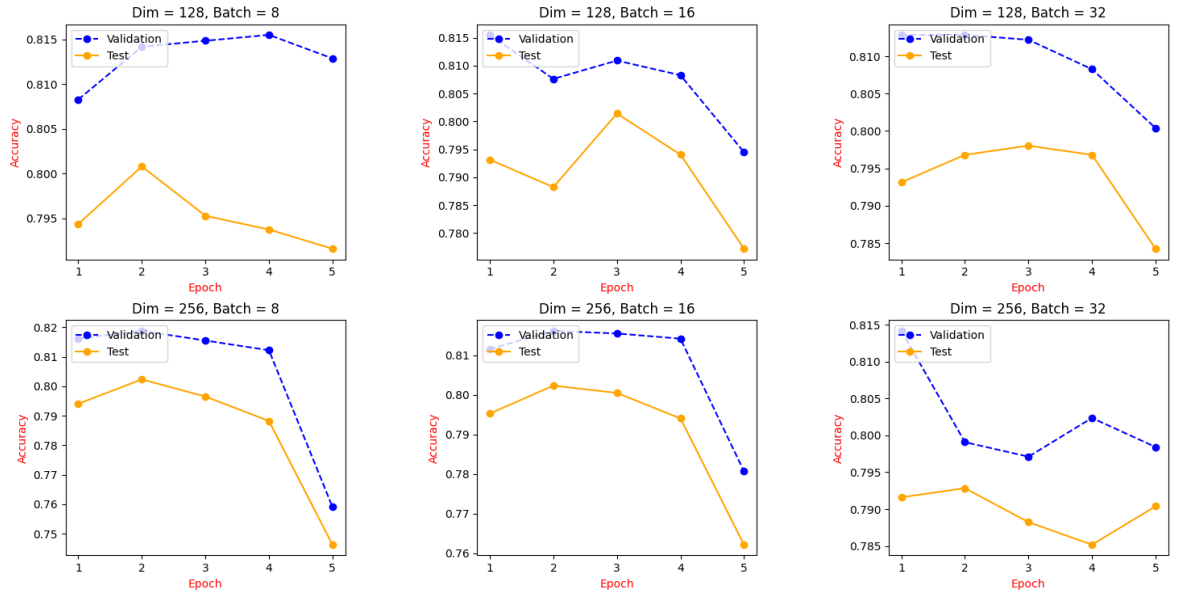
4.4.2 Result of Neural Network

The graph below is the accuracy of validation and test data per epoch with different embedding dimensions and batch size.



4.4.3 Result of Siamese Network

The graph below is the accuracy of validation and test data per epoch with different embedding dimensions and batch size.



5 Conclusion

Based on the previous several experiments, we can conclude that different models get similar results and they are not bad. There are still some future works that need to be done to get a better solution to this important problem.

1. Texts of other languages.
2. Take in more information e.g. keyword, location.
3. Stricter rules for text cleaning.
4. Different means of tokenization.
5. Better way to represent a token.
6. Find the best set of hyperparameters.

If we can further improve the overall performance, especially the recall (TP/(TP+FN)), we can detect almost all the texts about disasters. Agencies(e.g. disaster relief organizations and news agencies) can use the model to programmatically monitor Twitter or other social media. They can be aware of a disaster immediately once the model sends out an alarm.

6 References

[1] *Natural Language Processing with Disaster Tweets*. (2020). Retrieved from https://www.kaggle.com/c/nlp-getting-started/overview.

[2] Thilina Rajapakse. (2020). *Understanding ELECTRA and Training an ELECTRA Language Model*. Retrieved from https://towardsdatascience.com/understanding-electra-and-training-an-electra-language-model-3d33e3a9660d.

[3] Kevin Clark, Minh-Thang Luong, Quoc V. Le & Christopher D. Manning. (2020, Apr). *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. Paper presented at ICLR 2020, Addis Ababa, Ethiopia.

[4] Gunes Evitan. (2020). *NLP with Disaster Tweets - EDA, Cleaning and BERT*. Retrieved from https://www.kaggle.com/gunesevitan/nlp-with-disaster-tweets-eda-cleaning-and-bert#4.-Embeddings-and-Text-Cleaning.

[5] Yossawadee Promwong. (2021). *Disaster-and-NonDisaster-tweets*. Retrieved from https://www.kaggle.com/yossawadeepromwong/disaster-and-nondisaster-tweets/output.

[6] Arnab Dey. (2021). *Disaster Tweets Prediction using NLP*. Retrieved from https://www.kaggle.com/arnab132/disaster-tweets-prediction-using-nlp.

[7] The Hugging Face Team. (2021). *Tokenizer*. Retrieved from https://huggingface.co/transformers/main_classes/tokenizer.html.

[8] The Hugging Face Team. (2021). *BERT*. Retrieved from https://huggingface.co/transformers/model_doc/bert.html.

[9] Mazi Boustani. (2021). *Detecting Disaster from Tweets (Classical ML and LSTM Approach)*. Retrieved from https://towardsdatascience.com/detecting-disaster-from-tweets-classical-ml-and-lstm-approach-4566871af5f7.

[10] *Gradient boosting*. (2021). Retrieved from https://en.wikipedia.org/wiki/Gradient_boosting.

[11] *Chapter 12 Gradient Boosting*. (2021). Retrieved from https://bradleyboehmke.github.io/HOML/gbm.html.

[12] *1.1. Linear Models*. (2021). Retrieved from https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression.

[13] *sklearn.linear_model.RidgeClassifier*. (2021). Retrieved from

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifier.html.
[14] *sklearn.linear_model.Ridge*. (2021). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html.
[15] Peter Franek. (2018). *What does sklearn "RidgeClassifier" do?* Retrieved from https://stackoverflow.com/questions/53911663/what-does-sklearn-ridgeclassifier-do.
[16] The Hugging Face Team. (2021). *Glossary*. Retrieved from https://huggingface.co/transformers/glossary.html#input-ids.
[17] Algorithmia. (2016). *What is natural language processing? Introduction to NLP*. Retrieved from https://algorithmia.com/blog/introduction-natural-language-processing-nlp.
[18] NLTK Project. (2021). *NLTK 3.6.2 documentation*. Retrieved from http://www.nltk.org.
[19] *sklearn.feature_extraction.text.TfidfVectorizer*. (2021). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.
[20] *Fine-tuning a BERT model*. (2021). Retrieved from https://www.tensorflow.org/official_models/fine_tuning_bert.
[21] LeeMeng. (2018). 進入 *NLP* 世界的最佳橋樑:寫給所有人的自然語言處理與深度學習入門指南. Retrieved from https://leemeng.tw/shortest-path-to-the-nlp-world-a-gentle-guide-of-natural-language-processing-and-deep-learning-for-everyone.html#%E4%B8%80%E5%80%8B%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF%EF%BC%8C%E5%85%A9%E5%80%8B%E6%96%B0%E8%81%9E%E6%A8%99%E9%A1%8C.
[22] Christopher Olah. (2015). Understanding LSTM Networks. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.
[23] allenlu2007. (2021). *Siamese Network* 孿生連體神經網絡: 簡單神奇的結構. Retrieved from https://allenlu2007.wordpress.com/2019/05/30/siamese-network-%E5%AD%BF%E7%94%9F%E7%A5%9E%E7%B6%93%E7%B6%B2%E7%B5%A1%EF%BC%9A-%E7%B0%A1%E5%96%AE%E7%A5%9E%E5%A5%87%E7%9A%84%E7%B5%90%E6%A7%8B/.
[24] Girish Keshav Palshikar, Manoj Apte & Deepak Pandita. (2017). *Weakly Supervised Classification of Tweets for Disaster Management.* TCS Research, Tata Consultancy Services Limited, 54B Hadapsar Industrial Estate, Pune 411013, India. Retrieved from http://ceur-ws.org/Vol-1832/SMERP_2017_peer_review_paper_1.pdf.
[25] Text Preprocessing. (2021) Retrieved from https://keras.io/zh/preprocessing/text/.
[26] Yanwei Liu. (2019). *Python*深度學習筆記*(*五*)*:使用*NLTK*進行自然語言處理. Retrieved from https://yanwei-liu.medium.com/python%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92%E7%AD%86%E8%A8%98-%E4%BA%94-%E4%BD%BF%E7%94%A8nltk%E9%80%B2%E8%A1%8C%E8%87%AA%E7%84%B6%E8%AA%9E%E8%A8%80%E8%99%95%E7%90%86-24fba36f3896.
[27] Abhinav Kumar, Jyoti Prakash Singh, Sunil Saumya. (2019). *A Comparative Analysis of Machine Learning Technique for DIsaster-Related Tweet Classification*. IEEE Region 10 Humanitarian Technology Conference, Depok, Indonesia. Retrieved from https://ieeexplore.ieee.org/document/9042443