Github repository: <https://github.com/871234342/NYCU_AICap_FinalProject>

[**Natural Language Processing with Disaster Tweets**](https://www.kaggle.com/competitions/nlp-getting-started)

The project aims to tackle the challenge of finding tweets about disaster. The ubiquitousness of smartphones allows people to share information in real-time, like the disaster they are observing. As a result, programmatically monitoring social media, like twitter, could help gathering information.

**Dataset**

The dataset is split into training and testing set. The training set contains twitter messages and their targets, indicating a disaster tweet or not, while the testing set contains only messages. There are 7613 samples for training and 3263 for testing.

**Preprocessing**

Different from written article or spoken language, the messages may have twitter handle, http URL, hashtags, and some special characters, which need to be removed. Based on their pattern in the messages (like hashtags always start with #), we can simply remove them by looking for the pattern.

Words like ‘the’, ‘is’, and ‘me’ (or stop wards) appear commonly but carry little information, removing them could help our model focus on those informative words. To do that, we utilize the stop word list form NLTK and remove the words in the list.

In English, a word may have different spelling to under different circumstances. For example, ‘speaking’ and ‘speak’, ‘am’ and ‘are’, ‘car’ and ‘cars’. Despite the difference, they express the same semantic. Converting the words back to their original form would have positive effect on the performance.

We transform the cleaned, lemmatized sentence into a vector by words counts, i.e., how many times a word shows up in a sentence. We then use tf-idf to find some potentially important words and emphasize them. The preprocessing is applied to both training data and testing data.

**Models**

**Multinomial Naive Bayes**

Since multinomial Naive Bayes classifier is suitable for classification with discrete features, I think words counts for text classification, which is our task, is great application for this algorithm. And to my surprise, this model performs better than my expectation. Although it only has 70 percent accuracy on training data, it actually took under 10 seconds to train this model. Furthermore, it gets 72 percent accuracy on testing data.

**Random forest classifier**

When it comes to simple but robust models, I think decision tree and random forest are the classic, especially for classification task. We can think of potentially important words as features of the classification task. And to my experience, most of the time random forest overshadow decision tree, which, in my opinion, is because random forest is a group of decision trees, and sometimes one decision tree might not be able to perform well enough. Sadly, it only gets 68 percent accuracy on training data but took almost 5 minutes. However, it surprisingly gets 73 percent accuracy on testing data. I guess it’s because on how training data and testing data are distributed and a bit of luck.

**Logistic Regression classifier**