GitHub repo: https://github.com/Rick-Feng-u/Case-Study-Disaster-Tweets
I got some inspiration from:
https://towardsdatascience.com/sentiment-analysis-predicting-whether-a-tweet-is-about-a-disaster-c004d09d7245

To see data analysis results run <EDA.py>
To see validation results run <train_and_validation.py>
To generate submission CSV for test data run <train_and_test.py>

Data Analysis:

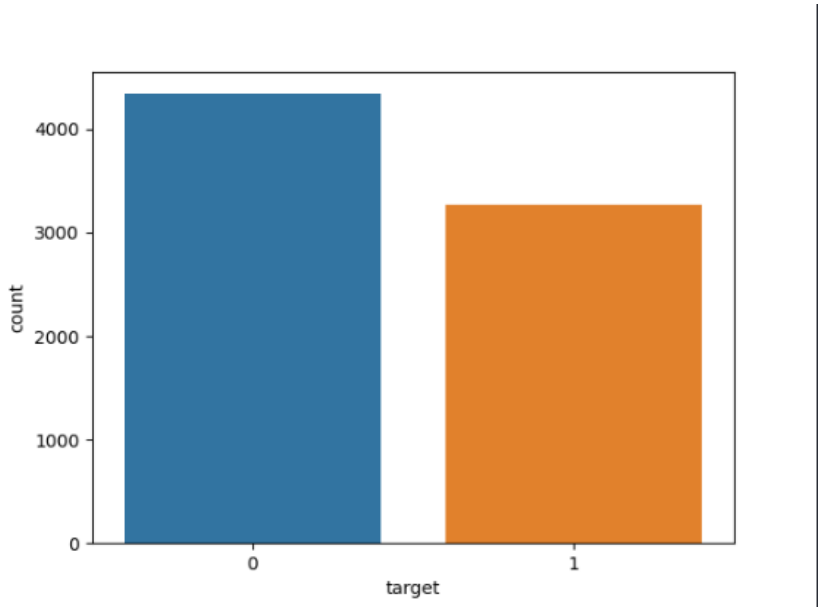
Before any work can be done the first thing is to figure out what kind of data I am working with so the file EDA.py(Exploratory data analysis) is used to do this

This is a very simple maybe a bit naive EDA but it gives me a clear picture of what I can do with the Data
When I first print out the training data

```
   id keyword location                                              text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M...
1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
2   5     NaN      NaN  All residents asked to 'shelter in place' are ...
3   6     NaN      NaN  13,000 people receive #wildfires evacuation or...
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as ...
```

I realize that column ‘text’ will be the main feature my ML model will take. This column contains the most information. Also, I quickly realize that there is a lot of NaN for ‘keyword’ and ‘location’. From further exploration, this is not true for keywords where many rows contain keywords but this is not the case for location. Location is missing 33% of the entire dataset. There are two options where 1) I can fill all the NaN with data, either finding every tweet and seeing their user location or some random data. However, this might make the dataset noisy. Therefore I chose not to include location as an input feature.
For keyword however is very much correlated with the target so I end up appending the keyword and text so the keyword can enhance its signal during training.

The distribution of the dataset is a bit concerning. There are a total of 7913 data points, 4342 are non-disaster and only 3271 are disaster tweets.



This is something called an imbalance of data. The sample distribution for minority dataset is 42%, just over the mild level (https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data)

Degree of imbalance	Proportion of Minority Class
Mild	20-40% of the data set
Moderate	1-20% of the data set
Extreme	<1% of the data set

This should be fine and does not require upsampling or downsampling, but this can be tested later to see if it will improve the accuracy of the model

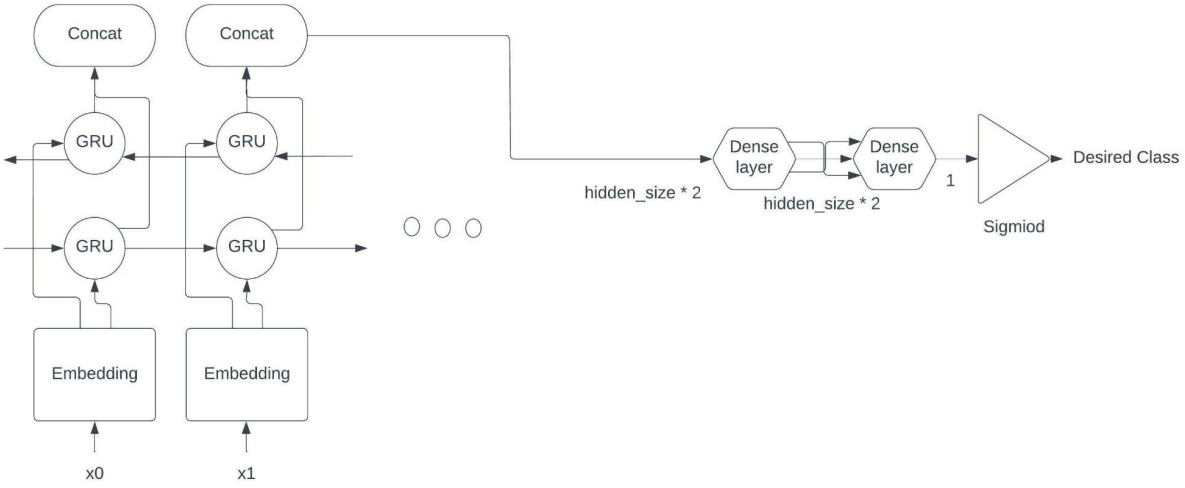
Preprocessing

When text is just being passed in it's at a raw state. This is where the data preprocessing comes in. All text will remove all special characters, emojis, and hyperlinks. Some words are not necessary for the model to learn so all cleaned text will be removed with stop words and stemmed. After all those processes, the text will be tokenized to integers so it can be a proper input for the model

When cleaned texts are converted into tokens they become lists. This will directly affect one thing. All texts have different lengths. Therefore, before it will become model input, we need to pad them. I append zeros at the end of all shorter lists so they are as long as the longest list. Zero is reserved for this exact reason(0 does not represent any word).

Model

I chose a bidirectional RNN model for this binary classification task. Specifically, I chose GRU or Gated Recurrent Unit. It is better than LSTM because it is as effective as LSTM but it is less performance heavy and computationally less intensive. I know Transfomer models such as BERT will perform better but considering the time limit and lack of GPU I oped with the bidirectional RNN model. This type of model is pretty decent at classification often berating trepidation ML models such as SVM or SGD Here is the general structure of the model:



The overall idea is bidirectional RNN is very similar to putting two undirected RNNs together but they are forwarding the ht in opposite directions. We concatenate the output of two RNNs for the corresponding output(in this case hidden state). Then forward it to two dense layers and output it to a sigmoid function that will produce a predicted target

Model evaluation

The loss function used the Binary Cross Entropy loss function and the optimizer is Adam(frankly Adam is pretty good for most of the task). I performed validation on the model. Train_test_split the training data where test_split = 0.2 and shuffle=True. The average accuracy of the model over 10 different validation testing is 74.8%. This is decent accuracy considering the accuracy of SVM based model according to this accuracy list:

	model_name	fold_1	fold_2	fold_3	fold_4	fold_5	avg
0	svm_count_vect	0.726481	0.727915	0.747116	0.748252	0.725835	73.511976
1	svm_tfidf_vect	0.711312	0.702304	0.718750	0.724138	0.698994	71.109958
2	logistic_regression_count_vect	0.737531	0.735802	0.750422	0.742998	0.750000	74.335061
3	logistic_regression_tfidf_vect	0.735371	0.713781	0.736748	0.738707	0.723063	72.953393
4	naive_bayes_count_vect	0.753165	0.736508	0.753036	0.755842	0.741281	74.796648
5	naive_bayes_tfidf_vect	0.737747	0.703030	0.739629	0.739774	0.720070	72.805019
6	SGD_count_vect	0.725567	0.713154	0.735818	0.724638	0.711726	72.218062
7	SGD_tfidf_vect	0.734930	0.727569	0.754371	0.744186	0.738513	73.991390
8	random_forest_count_vect	0.727578	0.726968	0.748528	0.731747	0.715736	73.011145
9	random_forest_tfidf_vect	0.712702	0.710169	0.739655	0.733837	0.711835	72.163992
10	lgbm_count_vect	0.720528	0.682310	0.719586	0.717735	0.711617	71.035515
11	lgbm_tfidf_vect	0.695427	0.687225	0.721368	0.711489	0.695580	70.221766
12	lstm_word_2_vec	0.752506	0.735000	0.749565	0.759839	0.755880	75.055806
13	lgbm_word_2_vec	0.564236	0.520354	0.551978	0.534283	0.547515	54.367329

(credit: Kurtis Pykes, URL: https://towardsdatascience.com/sentiment-analysis-predicting-whether-a-tweet-is-about-a-disaster-c004d09d7245)

There are some potential ways to increase the accuracy such as giving model L1 or L2 regulation, adjusting the learning rate, and putting an early stop in the training function so it prevents some overfitting(numer of epochs = 2 seems to be a sweet spot, and early stop did minor improvement) or in the data analysis section upsampling or downsampling the imbalanced data. Currently, the only thing I did to improve model accuracy is adding dropout where p=0.2 after bidirectional GRU and after the first Dense layer.