UNIVERSITY OF MARYLAND

Neural Network Project

Natural Language Processing with Disaster Tweets

Prepared by

Keshav Gupta, Chidubem Nwabunze, Saimah Khan, Jasmine Hill, Allika Thadishetty

Kaggle Team Name

BUDT737_Spring23_22

CONTENTS

Introduction

The natural language processing with disaster tweets is a Kaggle competition challenging participants to create a machine learning code that can accurately classify whether or not a tweet is a real disaster. The provided datasets consist of thousands of tweets, with each tweet labeled as either "disaster" (target = 1) or "not a disaster" (target = 0). We are given a training dataset with labels and a test dataset without labels.

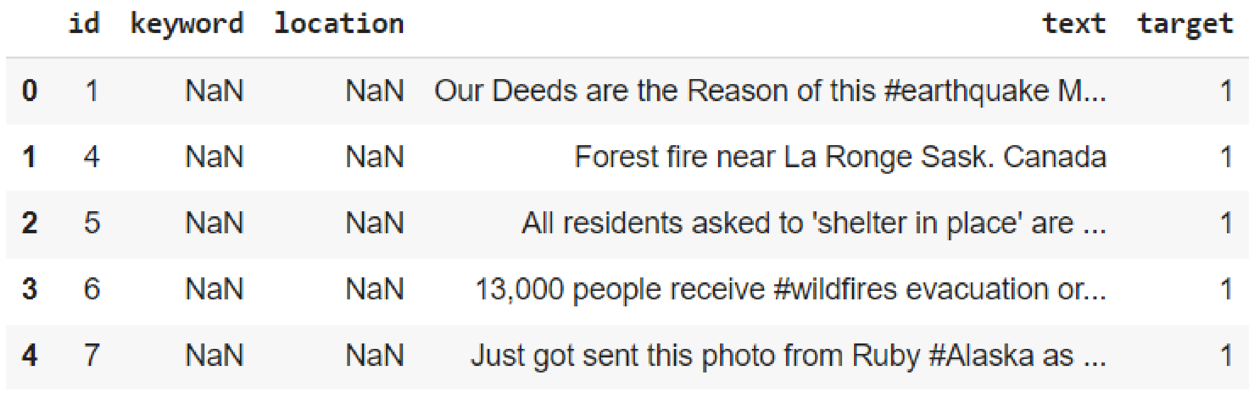
In our project, we utilize natural language processing (NLP) techniques such as text preprocessing, tokenization, and text vectorization to transform raw text data into vectors for our Machine Learning algorithm. We attempt to achieve the highest training accuracy since we are not provided the test labels to examine our model’s accuracy when predicting the test data.

As we created our model from scratch, we will not have any Kaggle code used as a reference. We will explain the methodology for our model later on in this report.

EDA and Data Preprocessing

EDA

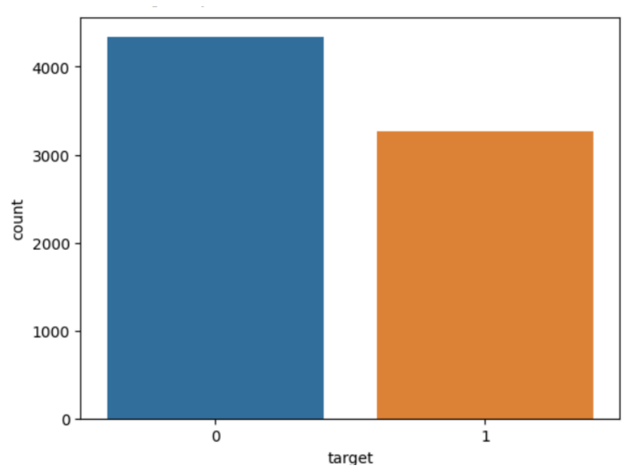
After importing our training data and the necessary packages, our first step is to print the head of the data to see a sample of the tweets.

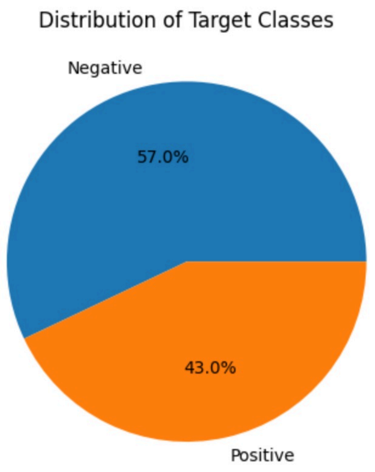


	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent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

As seen above, the keyword and location columns contained many null values and the text entries had punctuation marks, symbols, and a combination of uppercase and lowercase letters.

We also analyzed the distribution of disaster tweets vs. non-disaster tweets in the target column of our training set. We created seaborn plots to see if the data was unbalanced.





As we can see, there are slightly more non-disaster tweets compared to disaster tweets in our training data. The graph displayed shows that the data is not too unbalanced.

Data Preprocessing

Our data preparation steps included multiple steps in order to ensure the raw data can be read properly by our model. The first step was to import all the necessary packages as well as download the NLTK tokenizer package to recognize stop words within the tweets.

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

Next, we manipulated the raw data so that it was in a readable format for our model. To do so, we made several changes to the text which involved converting letters to lowercase, removing URLs, emojis, stopwords, special characters, and numbers. We do this because we want our algorithm to only focus on the aspects of the text with the most importance. We defined a function that performed all of these steps and then applied them to our training and test datasets.

```
# Preprocessing the text data
def preprocess_text(text):
    # Remove URLs
    text = re.sub(r"http\S+|www\S+|https\S+", "", text, flags=re.MULTILINE)

    # Remove emojis
    text = text.encode('ascii', 'ignore').decode('ascii')

    # Remove stopwords
    stopwords_set = set(stopwords.words("english"))
    text = " ".join([word for word in text.split() if word.lower() not in stopwords_set])

    # Remove special characters and numbers
    text = re.sub(r"[^\w\s]", "", text)
    text = re.sub(r"\d+", "", text)

    # Convert text to lowercase
    text = text.lower()

    return text
```

```
train_data['text'] = train_data['text'].apply(preprocess_text)
test_data['text'] = test_data['text'].apply(preprocess_text)
```

The results of our preprocessing steps resulted in the following output:

```
train_data.head()
```

	id	keyword	location	text	target
0	1	NaN	NaN	deeds reason earthquake may allah forgive us	1
1	4	NaN	NaN	forest fire near la ronge sask canada	1
2	5	NaN	NaN	residents asked shelter place notified officer...	1
3	6	NaN	NaN	people receive wildfires evacuation orders ca...	1
4	7	NaN	NaN	got sent photo ruby alaska smoke wildfires pou...	1

As you can see, the text column is free of symbols, numbers, stopwords, and more after applying our function.

Next, we will tokenize and pad the preprocessed text data. Tokenizing involves splitting text into individual tokens or words. Each token represents a meaningful unit of text. Tokenization is important for our analysis because it breaks down the text into discrete components so that further analysis and processing happen at the token level. The padding ensures that all text sequences have equal lengths, encouraging efficient batch processing and compatibility for the model.

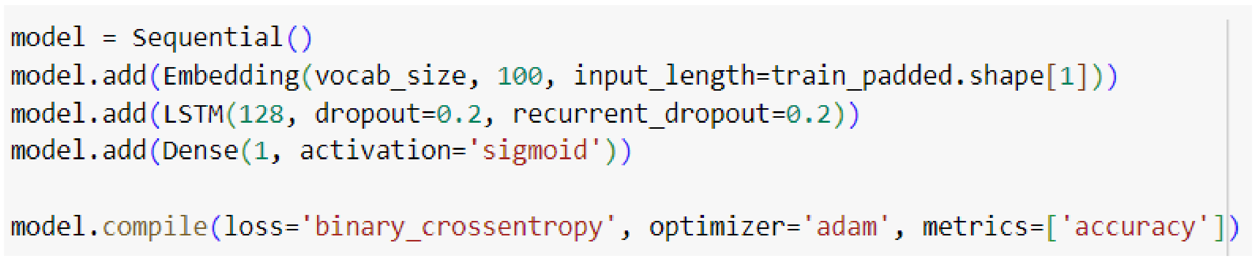
```
# Tokenization and padding
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data['text'])
vocab_size = len(tokenizer.word_index) + 1

train_sequences = tokenizer.texts_to_sequences(train_data['text'])
train_padded = pad_sequences(train_sequences)
```

We created a Tokenizer object and fit it on the preprocessed text column of the training data using the fit_on_texts function. Then, we get the size of the vocabulary by adding 1 to the length of the tokenizer's word index dictionary. Next, it converts each preprocessed text string in the text column of the training data into a sequence of integers using the texts_to_sequences method. Finally, it pads each sequence with zeros so that they are all of equal length using the pad_sequences method.

Now our data is ready to be used in our model.

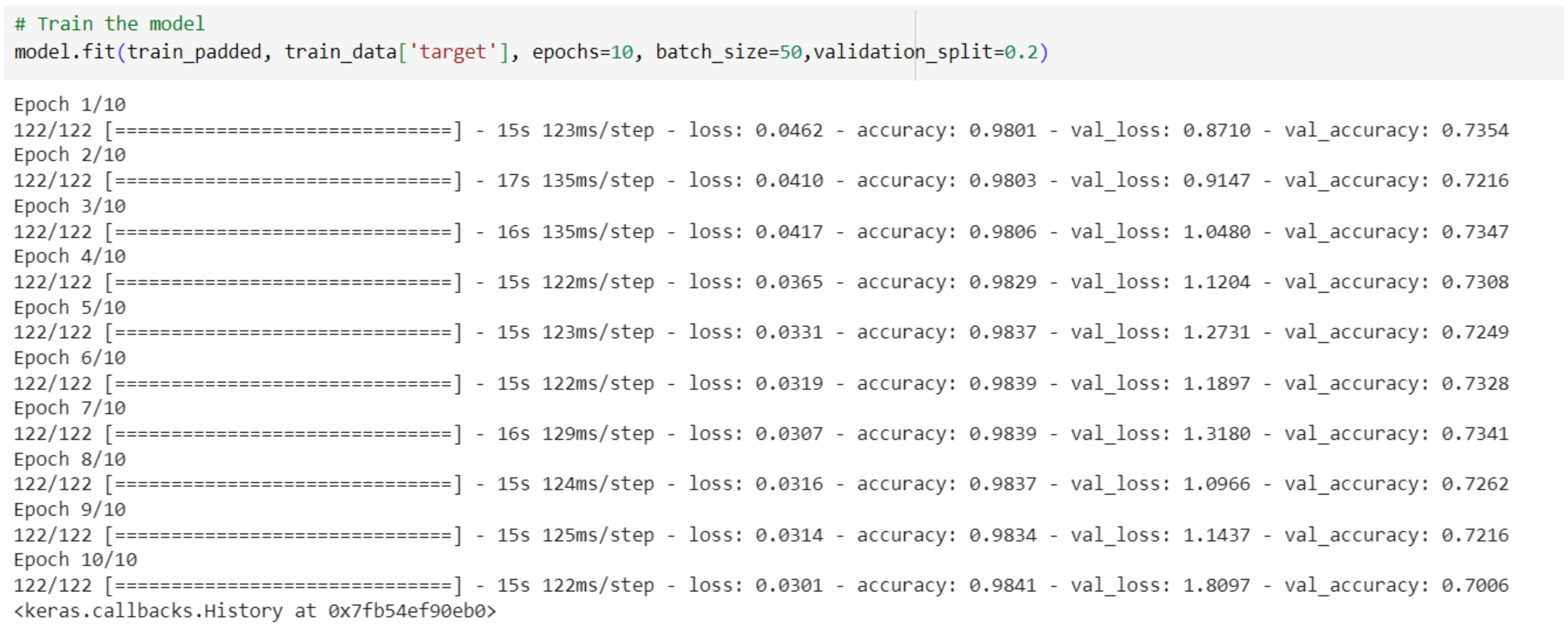
Methodology

We decided to create a model from scratch to apply natural language processing. We created a sequential model using Keras' Sequential function. We then added an embedding layer that maps each integer in our sequences to a dense vector of a fixed size of 100. Then, we added an LSTM layer with 128 units and a dropout rate of 0.2 to prevent overfitting. Finally, we added a dense output layer with one unit and a sigmoid activation function to produce binary classification predictions. After this, we compiled the model with a binary cross-entropy loss function, Adam optimizer, and accuracy metric.



```
model = Sequential()
model.add(Embedding(vocab_size, 100, input_length=train_padded.shape[1]))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

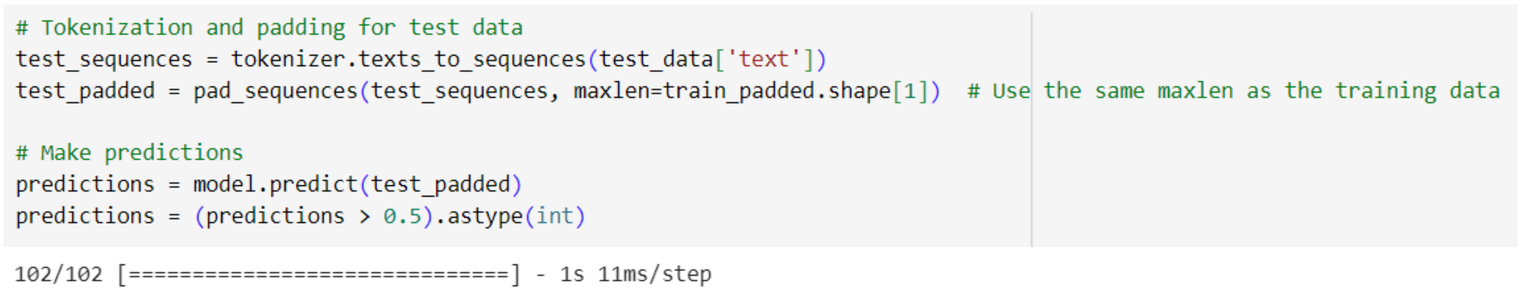
Our code fitted the compiled model on the padded training data with 10 epochs, a batch size of 50, and a validation split of 0.2. The fit() function trains the model on the training data and evaluates it on the validation data at the end of each epoch. We can see the training progress printed during training below:



```
# Train the model
model.fit(train_padded, train_data['target'], epochs=10, batch_size=50,validation_split=0.2)
```

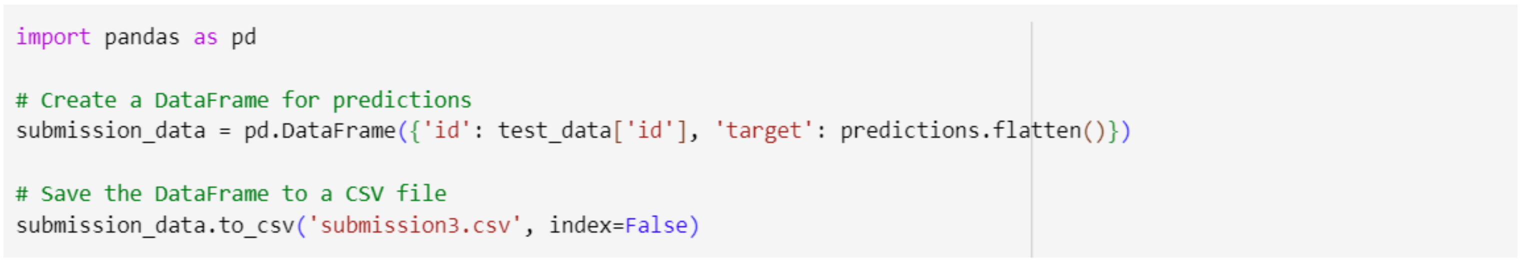
```
Epoch 1/10
122/122 [==============================] - 15s 123ms/step - loss: 0.0462 - accuracy: 0.9801 - val_loss: 0.8710 - val_accuracy: 0.7354
Epoch 2/10
122/122 [==============================] - 17s 135ms/step - loss: 0.0410 - accuracy: 0.9803 - val_loss: 0.9147 - val_accuracy: 0.7216
Epoch 3/10
122/122 [==============================] - 16s 135ms/step - loss: 0.0417 - accuracy: 0.9806 - val_loss: 1.0480 - val_accuracy: 0.7347
Epoch 4/10
122/122 [==============================] - 15s 122ms/step - loss: 0.0365 - accuracy: 0.9829 - val_loss: 1.1204 - val_accuracy: 0.7308
Epoch 5/10
122/122 [==============================] - 15s 123ms/step - loss: 0.0331 - accuracy: 0.9837 - val_loss: 1.2731 - val_accuracy: 0.7249
Epoch 6/10
122/122 [==============================] - 15s 122ms/step - loss: 0.0319 - accuracy: 0.9839 - val_loss: 1.1897 - val_accuracy: 0.7328
Epoch 7/10
122/122 [==============================] - 16s 129ms/step - loss: 0.0307 - accuracy: 0.9839 - val_loss: 1.3180 - val_accuracy: 0.7341
Epoch 8/10
122/122 [==============================] - 15s 124ms/step - loss: 0.0316 - accuracy: 0.9837 - val_loss: 1.0966 - val_accuracy: 0.7262
Epoch 9/10
122/122 [==============================] - 15s 125ms/step - loss: 0.0314 - accuracy: 0.9834 - val_loss: 1.1437 - val_accuracy: 0.7216
Epoch 10/10
122/122 [==============================] - 15s 122ms/step - loss: 0.0301 - accuracy: 0.9841 - val_loss: 1.8097 - val_accuracy: 0.7006
<keras.callbacks.History at 0x7fb54ef90eb0>
```

Next, we will prepare our test data for submission. As we did with the training data, we will tokenize and pad the test data so that it is readable to our model. Then, we will use our model to predict the test labels (target = 0 or target = 1) and create a data frame to add the test results to the submission file.



```
# Tokenization and padding for test data
test_sequences = tokenizer.texts_to_sequences(test_data['text'])
test_padded = pad_sequences(test_sequences, maxlen=train_padded.shape[1])  # Use the same maxlen as the training data

# Make predictions
predictions = model.predict(test_padded)
predictions = (predictions > 0.5).astype(int)
```

```
102/102 [==============================] - 1s 11ms/step
```

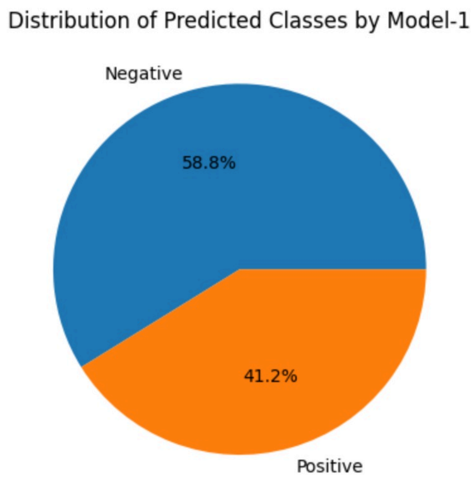


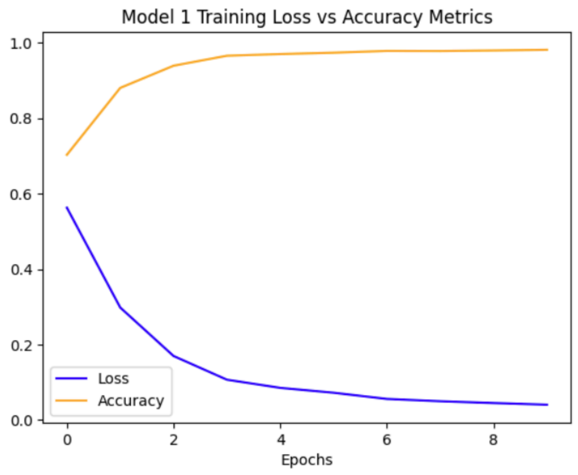
```
import pandas as pd

# Create a DataFrame for predictions
submission_data = pd.DataFrame({'id': test_data['id'], 'target': predictions.flatten()})

# Save the DataFrame to a CSV file
submission_data.to_csv('submission3.csv', index=False)
```

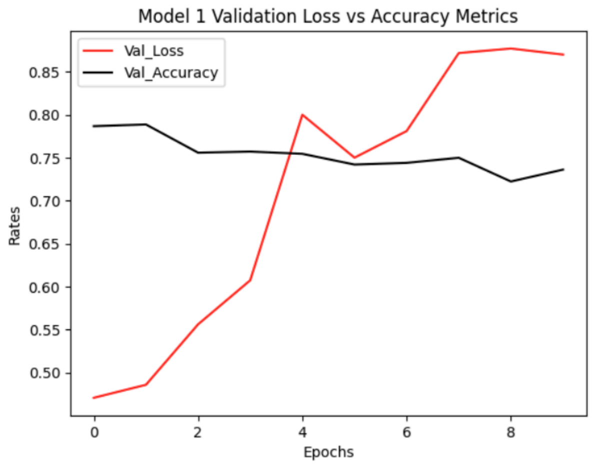
Results and Reports

Model-1

In the model we used for the Kaggle competition, the visualizations below display our test prediction results:





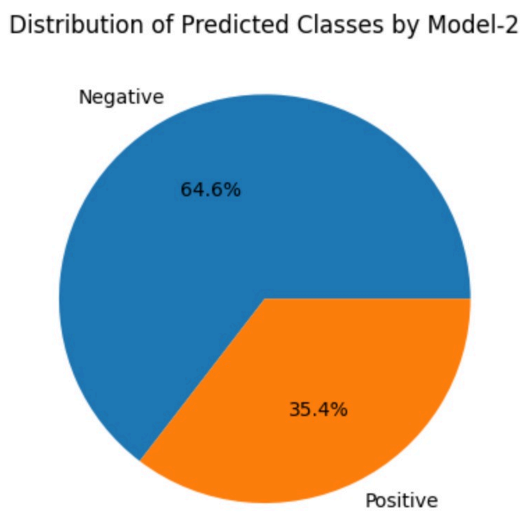


Model-1 Kaggle Score

Upon submission to the Kaggle competition, our model received a score of

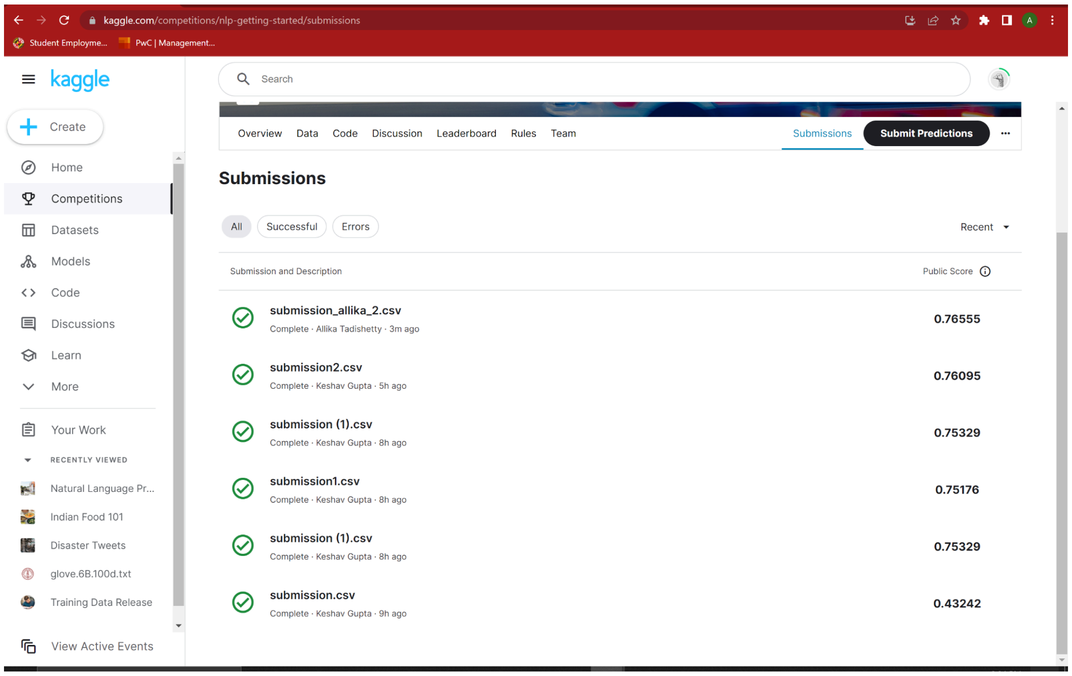
Model-2

We were also considering using a different model, so we generated the test prediction results of this model as well (Model-2) in order to compare the models prior to selecting Model-1 (which this report has been based on). Below are the test prediction results of Model 2:



Even though we had similar accuracies and results from both models, we decided to move ahead with Model-1. Based on the visualizations, we can see that Model-2 has more type-2 errors (the model tends to label more tweets as not related to an actual disaster when it actually does relate to a natural disaster), which is dangerous here in case of predicting with new, unfamiliar data (the test dataset).

KAGGLE SCORE



Submissions

Submission and Description	Public Score
submission_allika_2.csv Complete · Allika Tadishetty · 3m ago	0.76555
submission2.csv Complete · Keshav Gupta · 5h ago	0.76095
submission (1).csv Complete · Keshav Gupta · 8h ago	0.75329
submission1.csv Complete · Keshav Gupta · 8h ago	0.75176
submission (1).csv Complete · Keshav Gupta · 8h ago	0.75329
submission.csv Complete · Keshav Gupta · 9h ago	0.43242

Individual Contributions and Takeaways

Each member had an individual responsibility to create their own machine learning code and then compare before choosing one model to enter into the Kaggle competition. Further breakdown of each member's contribution is stated below:

Keshav Gupta

I created the model using the LSTM technique and used the NLTK library to remove the stopwords from the data. I used various combinations of loss functions, optimizers, batch size, validation split, and epochs and came up with the best model.
I learned about various loss functions, the NLTK library, teamwork, tokenizing, and padding the text. I learned about the importance of loss functions, batch size, and epochs. I understood when the model started overfitting the training set and how to avoid it.

Chidubem Nwabunze

I created a model based on one of those samples shared by the Professor. I modified the group's data pre-processing, modified the codes, implemented some model adjustments, and then generated the outcome which I shared with the team. During meetings, I contributed to discussions and helped in steering the group toward the successful completion of our project. I also created visualizations showcasing the accuracy and loss rates of the primary model's training and validation sets.

I learned a few things while working on this project. First, I learned more about data preprocessing and how it improves the quality of the data and the performance of the model. I learned about tokenization, stop word removal, stemming, removing punctuation, lowercase, etc. I also learned about feature engineering, and the use of word embeddings like Glove to capture contextual information of the text. Model selection was another thing I learned. As we did during classes, I learned that the choice of the model is very crucial in achieving high accuracy. We experimented with different models and fine-tuned the models using different deep learning and neural networks, as well as different parameters, and packages, before deciding which model to adopt. The project also allowed me to collaborate more with my team members and learn from one another.

Saimah Khan

In the initial discussions I contributed to brainstorming the course of action to be followed in the project, and we decided on taking up the Neural Network model. Like all the team members, I too built my model in Keras improvising on the sample model shared by the Professor. I then participated in Model Evaluation and chose the best model based on the highest accuracy. I helped in entering the competition in Kaggle and building a team. We prepared the

Project Report after carefully including all the parameters advised in the project requirements. I actively attended and participated in all group meetings to timely submit the project.

This project taught me how to perform machine learning for text mining. I could understand and implement Data Pre-processing to make the raw data fit for use in the model. I learned how to remove Stop Words, remove punctuation, lowercasing the test, Tokenisation, etc. During the project, I explored feature engineering methods to extract relevant information and create meaningful representations from the raw text data. This involved techniques like extracting numerical features from the text, such as word count or average word length, as well as creating indicators for specific patterns or structures within the tweets. It was good exposure to collaborate with group members of different perspectives and achieve our target as a team.

Jasmine Hill

My contribution to the project was to work on one separate code, which was later used to compare amongst the rest of the group. My code was based on the Nlp tweets code, which used a Keras model. Once we decided on which model to use, I was then responsible for helping create and contribute to writing the report. I also referred back to the code and worked with the group to confirm our understanding of the code before finalizing my portion of the report.

This project has taught me several things regarding natural language processing (NLP). I learned what tokenizers are and what they are used for. A tokenizer is used to split the text into individual words (or tokens). This process converts the tweets into a form that is more understandable to a machine learning algorithm, hence its usage in our project. I also learned about text vectorization and its purpose. We used GloVe as a text vectorization, which converted the text into vectors (another method that converts the tweets into a more understandable format for the algorithm).

Allika Thadishetty

I provided the initial preprocessing and a neural network model. It was a learning and implementing experience. First, we started with text preprocessing: this involved preparing the data(tweets) in a form that can be used up by the neural networks to mine the sentiment. This involved various steps like removing URLs, emoticons, special characters, numbers, and English stop words(these do not contribute to the sentiment analysis of text), and converting the entire reformed text to lowercase. We achieved this using regular expressions.

Tokenization: I first tried to use the GloVe Embedding, which generates word embeddings based on co-occurrence statistics of words in a large corpus, but later on, decided not to move ahead with it as I thought of training our own word embeddings from scratch using the specific dataset. In this way, the word embeddings are learned as a part of the neural network model during the training process. I learned this is advantageous when we have a specific domain or dataset with unique characteristics. We went ahead with the Keras tokenizer, text_to_sequences method(converting text data into integer sequence), and pad_sequences(making all sequences equal in length)

Neural Network Model: As decided each one of us created a neural network model, out of which the one with the best results was chosen. My model, even though had similar accuracy as the chosen one, had to be left out because the type-2 error was more, in this case, the model was classifying the tweets that belonged to an actual natural disaster as the alternate. So we went ahead with the model presented above.

I created the report formatting and wrote the introduction data preprocessing and EDA portions of the report along with Jasmine Hill.

In the whole process I have learned to use the sequential API of Keras, I have also made use of several layers like the input layer(to specify input data shape), embedding layer(to learn to capture meaningful relations between words based on the usage in data), LSTM layers(to capture long-term dependencies and sequential patterns in input sequences), dropout layers(to prevent overfitting) and finally dense layers(to learn complex patterns and relations in input data) with a sigmoid activation function for binary classification. I then went ahead with the model fitting and tokenizing the test data for predictions.