BUDT737: Big Data and Artificial Intelligence

Team Members:
Srikar Alluri
Maria Shaikh
Aishwarya Sadagopan
Madathil Geetanjali Menon
Mingchen Feng
Team Name on Kaggle: BUDT737_Group19

INTRODUCTION

The growth of textual data from various sources, including social media, news articles, and consumer evaluations, has made it urgently necessary to analyze and comprehend this enormous volume of unstructured data in the modern digital era. For organizations, researchers, and decision-makers, the ability to glean useful insights from text data as well as identify its underlying sentiment and patterns has become increasingly important. Techniques for natural language processing (NLP) are used in this situation.

The code presented here focuses on harnessing the power of NLP to preprocess text data(in our case, tweets), employing a range of techniques to refine and enhance its quality. By removing irrelevant words known as stopwords, performing lemmatization to reduce words to their base form, and eliminating unwanted characters and symbols, we can obtain a cleaner and more manageable dataset. This preprocessing step is essential for improving the accuracy and effectiveness of downstream tasks such as sentiment analysis, classification, and information retrieval.

We then explore deep learning by training a model that blends Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN) architectures. This model learns to better reliably predict the target variable by utilizing the preprocessed text input. We want to increase the model's accuracy and dependability of predictions by training it on this expanded dataset, allowing for more informed decision-making based on the studied text data.

In conclusion, this code not only exemplifies the value of NLP methods for preparing text input but also shows how these methods may be easily incorporated with deep learning models. We may use these techniques to mine the immense sea of textual data that is now available for significant insights, patterns, and knowledge.

CODE EXAMPLE (reference code)

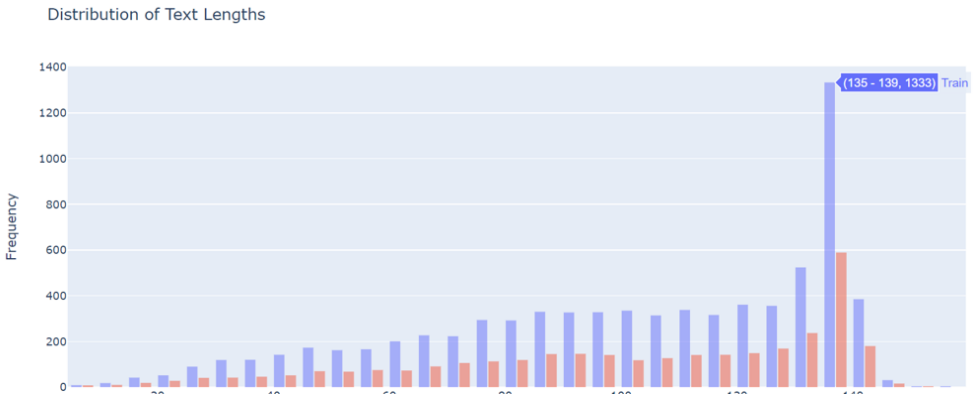
The link to the code that we have referenced from can be found here.

METHODS USED

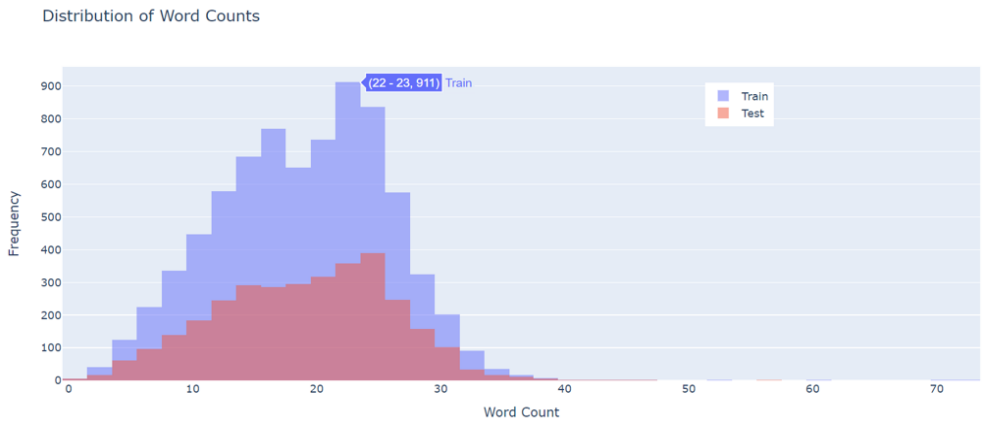
Data Exploration

Data exploration is an essential component of any data analysis project and it's main objective is to fully understand the dataset including its patterns and trends. In this project, we started the exploration by looking at the distribution of the target variable, which allowed us to understand the make-up of the dataset and identify the balance between the two groups. This analysis was essential for finding any potential class imbalances that needed to be taken into account in the next modeling phases. There does not seem to be class imbalance.

The exploration then focuses on the properties of the text data. It looks at the distribution of text lengths, which offers information on how tweet lengths vary. As it affects the complexity and structure of the data, understanding the range of text lengths can be crucial for feature engineering and model selection.



Finally we look at how often certain words appear in the text data. This explains the variety and richness of the vocabulary employed in the tweets. It assists in determining the variety of word counts that may be used to extract text-based characteristics or guide the choice of suitable modeling strategies.



Plotly : https://www.kaggle.com/code/kanncaa1/plotly-tutorial-for-beginners

We also performed an analysis of text complexity using the Flesch-Kincaid Grade Level metric. We found out the most and least complex string in the dataset, along with its corresponding target value and Flesch-Kincaid Grade Level. The purpose of this analysis was to provide insights into the complexity of the text data and understand how it relates to the target variable.

```
The most complex string in df_train, along with its target value and grade level:

#Pandemonium.iso psp http://t.co/HbpNFOAwII
Target value: 0
Flesch-Kincaid Grade Level: 32.8
-----------

The least complex string in df_train, along with its target value and grade level:

LOOOOOOL
Target value: 0
Flesch-Kincaid Grade Level: -3.5
-----------

Average grade level in df_train: 8.020267962695389
Average grade level in df_test: 8.063806313208703
```

We also wanted to compare the complexity of the training and test data to see if our model would perform well on the test data. The similarity of the average grade levels in df_train and df_test suggests that the two datasets have similar distributions in terms of the grade level of the texts. This indicates that the texts in the test dataset are representative of the texts in the training dataset in terms of their complexity and difficulty.
Having similar grade level distributions is desirable in many natural language processing tasks, as it ensures that the model trained on the training dataset can generalize well to the test dataset. If the grade level distributions were significantly different, it could indicate a mismatch between the training and test data, which may affect the performance of the model when applied to real-world scenarios. By having similar grade level distributions, it becomes more likely that the model trained on the training dataset will

be able to accurately predict and comprehend texts of similar complexity in the test dataset. This allows for a more reliable evaluation of the model's performance and its ability to handle texts at a specific grade level.

Flesch-Kincaid : https://www.kaggle.com/code/yhirakawa/textstat-how-to-evaluate-readability

Data Cleaning and Preparation

To enhance the quality and usefulness of the text data for modeling, the data cleaning and preparation step was carried out. Spacy, which specializes at processing massive amounts of text and carrying out tasks like named entity identification, was initially used for data cleansing. But because our project focused on thorough text processing and analysis, including stemming techniques, we chose NLTK. The wide support that NLTK offers for numerous linguistic aspects makes it a good fit for our particular needs.

We used a number of methods, including deleting HTML elements, URLs, and punctuation. After removing unnecessary elements, we performed essential text preprocessing using tools and libraries from the Natural Language Toolkit (NLTK). We employed tokenization to split the text into individual words using the word_tokenize() function.To refine the words further, we implemented lemmatization and stemming techniques. Lemmatization reduced words to their base form, while stemming aimed to reduce them to their root form, improving text consistency.To eliminate non-informative words, we removed stopwords from the NLTK stopwords corpus. These common words, such as "the" or "and," do not contribute significantly to text meaning and can introduce noise.Additionally, we removed non-alphabetic characters, ensuring that only meaningful alphabetic words remained in the text. This step eliminated numeric values, symbols, and special characters that might not be relevant for subsequent modeling and analysis tasks.

The unique words reduced from 31,924 to 12,968 which showcases the effectiveness of the cleaning process and highlights the improvements made in preparing the data for modeling.

Stemming and tokenization : https://www.geeksforgeeks.org/python-stemming-words-with-nltk/

Spacy : https://blog.quantinsti.com/spacy-python/

Early Stopback

We then implemented early stopping using the EarlyStopping callback from the TensorFlow Keras library. (https://keras.io/api/callbacks/early_stopping/). This is a technique used during model training to prevent overfitting and improve generalization by monitoring validation loss.If there is no improvement in the validation loss for a specified number of epochs (in this case, 15), training is stopped early. This technique helps optimize the model's performance while avoiding unnecessary computational resources.

OUR MODEL

In our model, we have improved the code from the original notebook by incorporating more advanced techniques for improving model performance and generalization. It includes random seed initialization for reproducibility, a flattening layer for reshaping the data, dynamically calculated hidden layer sizes, regularization for controlling overfitting, and a learning rate schedule for optimizing the learning process. These enhancements can potentially result in better model performance and improved ability to handle diverse data.

Initially, we planned to start the model with an LSTM (Long Short-Term Memory) layer, which is a popular choice for sequence data processing in natural language processing tasks. However, during the development process, it was observed that the performance of the LSTM layer was not optimal when directly applied to the input data. Upon further investigation, it was discovered that including an embedding layer before the LSTM greatly improved the model's performance. The embedding layer learns the dense representations of words that capture semantic relationships and contextual information, which can enhance the model's ability to extract meaningful features from the text data. By incorporating an embedding layer, the model became more effective in capturing the underlying patterns and nuances present in the tweets, resulting in improved prediction accuracy. After including the embedding layer, the LSTM layer was not used in the model. Instead, the model architecture was modified to include a flatten layer and dense layers following the embedding layer. This change was made based on the observation that the combination of the embedding layer and the subsequent flatten and dense layers produced better results compared to using an LSTM layer directly.

Some in-depth details about the other improvements made to the model are:

1. **Defining layer size**: The original code specifies the number of units in the 2 Dense layers as 256 and 128. We have changed this to a code that calculates the number of units in the input layer and hidden layers for a neural network model to be used for binary classification of tweets into real disaster or not. The code first calculates the shape of the input data and sets the number of output units to 1 since this is a binary classification problem. Then, it calculates the ratio of the cube root of the product of the input dimensions to the number of output units. This ratio is used to determine the number of units in the hidden layers. It is better to use shape_input to calculate the number of units in the input layer rather than directly using a specific number because it allows us to easily adjust the size of the input layer to fit the size of our input data. Using a fixed number could lead to suboptimal performance or even errors if the input data size changes. By using the shape of the input data to calculate the number of units in the input layer, we can ensure that the network is appropriately sized for the specific input data we are working with.
2. **Incorporating a learning rate schedule:** Our model, with the incorporation of a learning rate schedule using ExponentialDecay, has the advantage of dynamically adapting the learning rate during training compared to the first model, which uses a fixed learning rate through the RMSprop optimizer. This adaptability allows for improved convergence and optimization of the

model's parameters. The ExponentialDecay schedule gradually decreases the learning rate over time, which can be beneficial for tasks that require an initially higher learning rate for exploration and a smaller learning rate for fine-tuning and convergence. By fine-tuning the learning rate schedule hyperparameters, such as the initial learning rate, decay steps, decay rate, and staircase parameter, the learning process can be optimized for the specific task and dataset, potentially resulting in improved model performance. Overall, the inclusion of a learning rate schedule in the second model offers more flexibility, adaptability, and control over the learning process, leading to better optimization, faster convergence, and improved generalization.

3. **Applying L2 regularization** (https://keras.io/api/layers/regularizers/)**:** In the second code, L2 regularization is applied to the second hidden layer using the kernel_regularizer=l2(0.001) argument. L2 regularization, also known as weight decay, is a technique used to prevent overfitting in neural networks. When L2 regularization is applied, a penalty term is added to the loss function during training. This penalty term discourages the model from assigning excessively large weights to the parameters of the second hidden layer. The penalty term is calculated as the sum of the squared values of the weights, multiplied by a regularization coefficient (0.001 in this case). By including this penalty term, the model is encouraged to distribute the importance of different features more evenly and avoid relying too heavily on a subset of features. The effect of L2 regularization is to impose a constraint on the model's weights, encouraging them to stay relatively small. This can help prevent overfitting by reducing the model's complexity and improving its ability to generalize to unseen data. By penalizing large weights, L2 regularization acts as a form of regularization that balances the trade-off between fitting the training data well and avoiding overfitting. Overall, the addition of L2 regularization to the second hidden layer in the second code helps to control the complexity of the model, reduce overfitting, and improve generalization performance by discouraging excessively large weights.

4. **Using a Flatten Layer instead of a Global Average Pooling 1D Layer:**

Our model uses a Flatten layer that is included after the embedding layer. The purpose of this layer is to reshape the input data into a flat vector, which can be beneficial when working with sequential data. The Flatten layer collapses the dimensions of the tensor, combining all the elements into a single continuous vector. This is achieved by "flattening" the tensor along the dimensions, resulting in a 1D tensor with dimensions (batch_size, sequence_length * embedding_dim).
We chose to use the Flatten layer over a global average pooling 1D layer for several reasons

1. Preserving Sequential Information: The Flatten layer retains the sequential order of the input data. This can be important when the order of the sequence holds valuable information for the task at hand, such as sentiment analysis or language translation. In contrast, a global average pooling 1D layer discards the sequential information by summarizing the sequence into a fixed-length representation.
2. Handling Variable Sequence Lengths: The Flatten layer can handle variable sequence lengths effectively. It allows for input sequences of different lengths to be flattened into a common 1D representation. This flexibility can be crucial when working with datasets that contain sequences of varying lengths, as it avoids the need for padding or truncation.

3. Capturing Local Patterns: The Flatten layer captures local patterns and dependencies within the sequence. It preserves the fine-grained information at each time step, enabling the subsequent layers to learn from these patterns. This can be advantageous in tasks where short-term dependencies or local patterns are important, such as named entity recognition or part-of-speech tagging.

5. **FOR Loops:** The for loops in the given code are used to iterate over different combinations of hyperparameters to train and evaluate multiple models. Let's break down the purpose of each for loop and explain their importance:

1. Batch Size Loop: This loop iterates over the BATCH_SIZES list, which contains different batch sizes. Batch size determines the number of training examples processed before updating the model's weights. Training with different batch sizes allows us to observe the impact of batch size on model performance and find the optimal value that balances computational efficiency and model convergence.
2. Activation Function Loop: This loop iterates over the activation_functions list, which contains different activation functions. Activation functions introduce non-linearity to the model and play a crucial role in capturing complex relationships within the data. By trying different activation functions, we can assess their effect on the model's ability to learn and generalize patterns in the data.
3. Embedding Dimension Loop(https://keras.io/api/layers/core_layers/embedding/): This loop iterates over the embedding_dims range, which contains different embedding dimensions. The embedding dimension determines the size of the dense vector representation for each word or token in the input. Varying the embedding dimension allows us to explore different levels of representation granularity and find the optimal dimension that balances model complexity and information retention.

During each iteration of the nested for loops, a model is created using the create_model function with a specific combination of hyperparameters. The model is then trained on the training data (train_input and train_output) using the specified batch size, activation function, and embedding dimension. The validation data (val_input and val_output) is used to evaluate the model's performance.
The validation accuracy (val_accuracy) of each model is recorded in the accuracy_values list. Additionally, information about the model, including the model itself, batch size, activation function, and embedding dimension, is stored in the models list. This allows us to keep track of the models and their respective hyperparameters.
After all the iterations, we can examine the accuracy_values list to find the highest validation accuracy and identify the corresponding model in the models list. This helps us identify the best-performing model among the different combinations of hyperparameters.

SCORES FROM KAGGLE

The test accuracy score for our final submission on Kaggle is 0.73. It can be accessed by using our group name - BUDT737_Group19 on Kaggle by going to the leaderboard section.
https://www.kaggle.com/competitions/nlp-getting-started/leaderboard?#

RESULTS AND REPORT

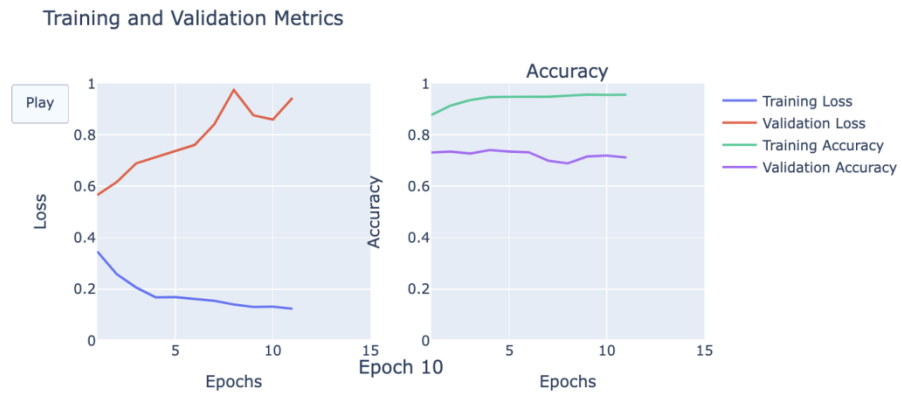
Our best performing model used 30 batches and elu activation function and 200 dimensions.
The validation accuracy achieved was : 0.7482.
After conducting an extensive hyperparameter search using a for loop, the optimal configuration for our text classification model was determined to be utilizing 30 batches, employing the ELU activation function, and employing a embedding dimensionality of 200. This configuration yielded the highest accuracy for our validation dataset.
To evaluate the performance of the model, we visualized the training and validation accuracies as well as losses and employed a confusion matrix to analyze the classification results and a ROC curve to understand the performance under classification thresholds.

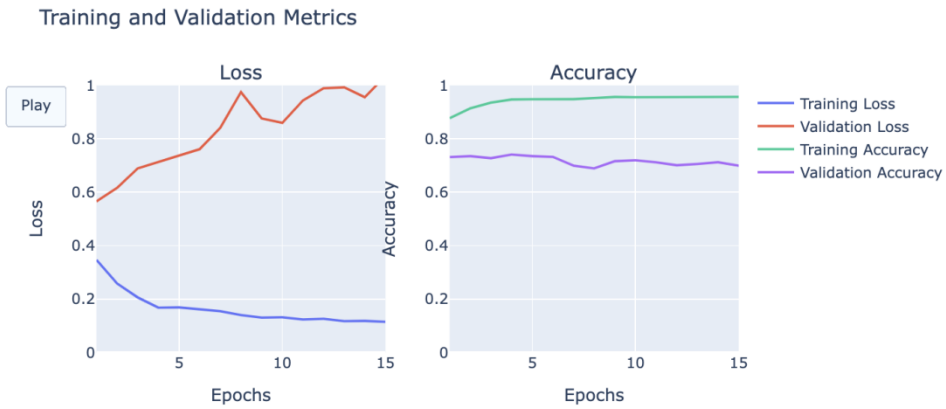
Training and Validation metrics:

Upon examining the graphs, we observe a discernible pattern wherein the training accuracy exhibits an upward trend as the number of epochs increases. This finding suggests that the model is effectively learning from the training data and improving its ability to classify text accurately over time.

Conversely, the validation accuracies display a mixed trend, indicating that the model's performance on unseen data varies across different epochs. Regarding the training loss, we note a consistent decreasing trend with an increase in the number of epochs. This pattern indicates that the model's learning process is gradually minimizing the discrepancy between predicted and actual class labels, resulting in improved accuracy over time.

.



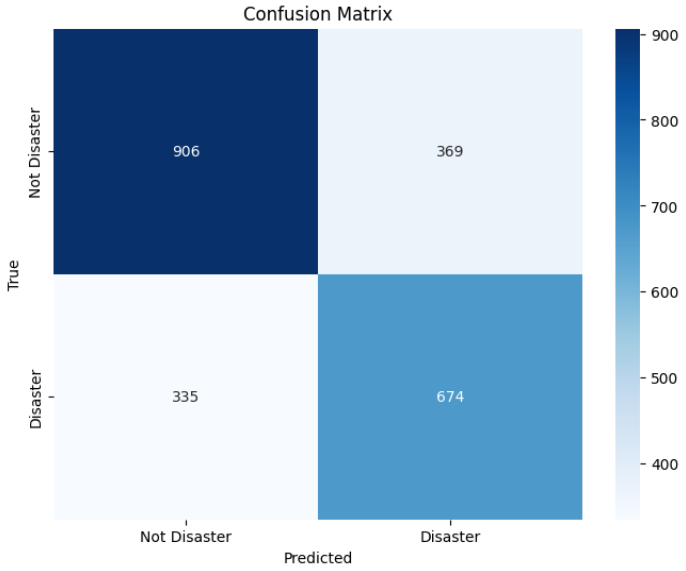


Confusion Matrix:

The confusion matrix, depicted in a plot below, allows us to assess the model's predictive capabilities by comparing the actual and predicted classes.

The True Positive Rate (TPR): The True Positive Rate also known as Sensitivity, indicates the proportion of actual "Disaster" instances that are correctly classified as "Disaster." In this case, the model achieves a TPR of 66.86% (674 / (674 + 335)), meaning that it correctly identifies about 66.86% of the actual "Disaster" instances.

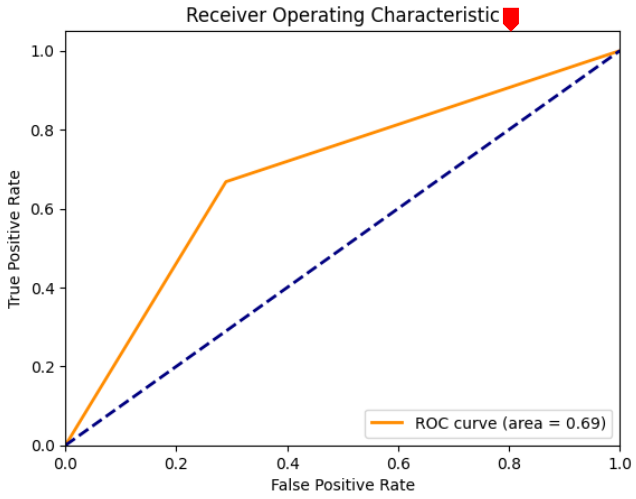
True Negative Rate (TNR): The True Negative Rate (TNR), also known as Specificity, represents the proportion of actual "Not Disaster" instances that are correctly classified as "Not Disaster." In this scenario, the model achieves a TNR of approximately 71.04%(906 / (906 + 369)), indicating that it correctly identifies about 71.04% of the actual "Not Disaster" instances.



ROC Curve:

ROC curve was plotted to gain insights into the model's discrimination ability. The ROC curve plots the true positive rate against the false positive rate at various classification thresholds.

The area under the ROC curve (AUC) is a single metric that summarizes the overall discrimination ability of the model. In this case, the AUC of 0.69 suggests that the model has a moderate level of discrimination ability in distinguishing between "Disaster" and "Not Disaster" instances. The fact that the model achieved an AUC of 0.69 is promising and suggests that the model demonstrates a moderate discrimination ability, surpassing a random classifier(AUC=0.5) and showing potential for distinguishing between "Disaster" and "Not Disaster" instances.



OUR CONTRIBUTION

Aishwarya Sadagopan:

During the project, I conducted the initial data exploration such as checking for potential data imbalance in target variable , distribution of text length ,distribution of word counts to gain insights into the patterns and trends present in the data. Additionally, I performed some important data cleaning procedures such as removing URLs and punctuations to ensure the data was in a suitable format for modeling. Finally, I created visualizations - Training vs Validation Accuracy and Loss to effectively represent the final output of the project.

Madathil Geetanjali Menon:
Enhanced the model's robustness and generalization abilities by including L2 regularization, which will allow for more precise predictions on unobserved data while reducing the danger of overfitting.

A crucial hyperparameter for deep learning models, the ideal embedding dimension, was also the subject of an iterative investigation I undertook. I have examined the model's accuracy across multiple embedding dimensions using an iterative loop. The embedding dimension that delivers the best accuracy has been identified through meticulous investigation, allowing the model to extract the most valuable semantic information from the text input.

Srikar Alluri:
I performed essential steps such as tokenization, and lemmatization on the text data. Tokenization allowed us to break down the text into individual words or tokens, enabling further analysis and processing. Additionally, I applied lemmatization, which reduced words to their base or root form, ensuring consistency and reducing variations in word usage.

I utilized GloVe embeddings to compare their performance with the embedding layer provided by the Keras framework. By experimenting with GloVe embeddings and comparing them with the Keras embedding layer, we aimed to identify the most effective embedding structure for our text classification model.

I conducted model evaluation on the validation data. I employed two widely-used evaluation techniques, the confusion matrix and the receiver operating characteristic (ROC) curve. This evaluation allowed us to assess the model's performance in terms of correctly identifying disaster and non-disaster texts.

Mingchen Feng
After researching for some information on the NLP problem, I noticed that the text preprocessing step is pretty important so I conduct some text preprocessing such as removing the website links, html flags, punctuation and emoji from the tweet.

Maria Shaikh

Since I wanted to explore our train and test data further before diving into the model, I contributed by implementing code to calculate the complexity of tweets using the Flesch-Kincaid Grade Level. By applying the textstat library, I computed the grade level for each tweet in the training and test datasets. This allowed me to identify the most and least complex tweets in the training set and determine the average grade level for both datasets. These calculations provided valuable insights into the linguistic characteristics of the tweets and their relationship to the target variable.
After conducting thorough research on the variables that affect the accuracy of NLP models, I worked on steps such as defining a ratio to calculate the value of input for hidden layers, creating for loops to see the difference between accuracies of different batch sizes and activation functions. I also replaced the optimizer from RMSprop to a dynamic optimizer as explained above.

WHAT EACH OF US LEARNT

Aishwarya Sadagopan:

The project helped me gain insights into data exploration. I also learned about data cleaning and visualizations using plotly for preparing the data. Additionally, I expanded my knowledge of Keras by experimenting with different layers and embeddings. This allowed me to understand their impact on the model's performance. Overall, the project enhanced my understanding of building and modifying neural network models, including the importance of layer selection and embedding choices in text classification tasks.

Madathil Geetanjali Menon:

Recognized the importance of regularizing the model through L2 regularization and fine-tuning the embedding dimension to maximize accuracy and capture crucial information from the input data.
I also realized how important it was to choose the right embedding dimension for the model. I thoroughly assessed the model's accuracy across various embedding dimensions using an iterative methodology. I was able to pinpoint the embedding dimension that produced the greatest accuracy thanks to my thorough analysis.

Srikar Alluri:

Throughout the project, we faced challenges with unclean datasets, selecting optimal hyperparameters, runtime complexities and interpreting the results. Addressing unclean datasets required various cleaning and processing techniques like stopword removal, lemmatization and so on that helped me understand various text mining methods. Selecting hyperparameters involved careful experimentation and tuning to achieve the best performance. Interpreting the results helped me understand various model evaluation techniques and helped in refining my approach.

Mingchen Feng

This project helped me learn about how the process of Natural Language goes. I also learned how I could cooperate with the regular expression to reach the goal of eliminating the unnecessary item from a sentence. I also learned that I could utilize the hyperparameter to pre-assign some values when doing the modeling.

Maria Shaikh

Through this project, I learned how to improve a machine learning model by incorporating various techniques. By making these improvements, I was able to achieve a higher accuracy and better generalization on my model compared to the initial implementation. Through the for loops in my code, I gained insights into the impact of different combinations of hyperparameters on the performance of the machine learning model. By iterating over various activation functions, embedding dimensions, and batch sizes, I learned how these choices influenced the model's ability to capture patterns in the data and make accurate predictions. Additionally, I learned how to use a learning rate schedule to dynamically adjust the learning rate during training, which helped optimize the model's convergence and overall performance. By evaluating and comparing the models based on their validation accuracy, I was able to identify the best-performing model and further refine my understanding of hyperparameter tuning.