Big Data Intelligence Project Report

Justinas Jučas (2024403399)
Department of Computer Science
Delft University of Technology, The Netherlands
Delft, 2628 CD
jst24@mails.tsinghua.edu.cn

William Yihao Zhang (2024403364)
Faculty IV - Electrical Engineering and Computer Science
Technical University Berlin, Germany
w.zhang@campus.tu-berlin.de

Gausse Mael Dongmo Kenfack (2024403346)
Department of Telecommunications, Services and Uses
INSA de Lyon, France
dmgs24@mails.tsinghua.edu.cn

Matthias Diederichsen (2024403452)
Faculty of Science, Major in Computer Science
Univeristy of British Columbia, Canada
dd24@mails.tsinghua.edu.cn

December 30, 2024

1 Introduction

1.1 Problem Explanation

In this competition, the goal was to build a machine learning model that can classify Tweets as either related to real-world disasters or not. Due to the informal and often ambiguous nature of social media language, distinguishing between genuine disaster-related posts and unrelated ones can be very difficult. For example, if we take two hypothetical Tweets "Forests are on fire" compared to "Lakers are on fire", the only difference lexically is the word "Forests" compared to"Lakers", yet as humans it's obvious that the former is talking about a forest fire while the latter is talking about a basketball team. For a machine, this is not so obvious, so in this notebook we will be exploring ways to deal with such issues and others.

1.2 Mindset and Approach

While training models can feel like a guessing game, our main approach to this problem was to gradually experiment and modify various models to grow in understanding of their underlying functionality and applicability to the problem. This approach helped us avoid the "black box" problem, as rather than throwing random inputs at the model, through many iterations we gradually improved our respective models, which gave us valuable insights. Since we were a group of 4, we had more manpower to try different approaches for both data preprocessing and model building, which will be discussed in detail in this report.

2 Data Analysis

2.1 Dataset Description

The dataset used for the project consisted of labeled samples for the binary classification task of identifying whether a tweet relates to a disaster (labeled as 1) or not (labeled as 0). The dataset comprised of 7613 labeled tweets for the training dataset and 3271 labeled tweets in the test dataset

Key Features consisted of:

1. id - a unique identifier for each tweet
2. text - the text of the tweet
3. location - the location the tweet was sent from (may be blank)
4. keyword - a particular keyword from the tweet (may be blank)
5. target, the label denoting whether a tweet is about a real disaster (only in train dataset)

2.2 Data Exploration

1. **Label distribution:** Non-disaster tweets account for 57% and disaster tweets for 43% of the data. Overall, the label distribution is relatively balanced. This distribution suggests that the dataset should not exhibit significant skewness or bias. However, it is still optimal to utilize weighted loss functions or metrics (e.g., F1-score) for the evaluation.
2. **Null Values:** In the training dataset, roughly 33.2% of entries had missing location data and roughly 33.8% was missing for the test set. This indicated that a significant portion of tweets do not include location information, which may limit its usefulness as a predictive feature. On the other hand, only 0.8% of keywords were missing for both train and test datasets. Given the low percentage of null values, the keyword feature remained reliable and could be leveraged for modelling.
3. **Keyword Analysis and Distribution** We analyzed the keyword column by plotting the top 20 most common keywords. Additionally, we examined the distribution of disaster and non-disaster tweets within each of these keywords. As observed in Figure 1, the keyword feature of the tweet often exhibited bias towards one label class over the other. For instance, the keyword *outbreak* is heavily associated with disaster tweets, while keywords such as *twister* or *hellfire* were mainly used in tweets not related to

disasters. Overall, this biased distribution of the keyword feature suggests that it may later serve as a meaningful predictor in the analysis. ,

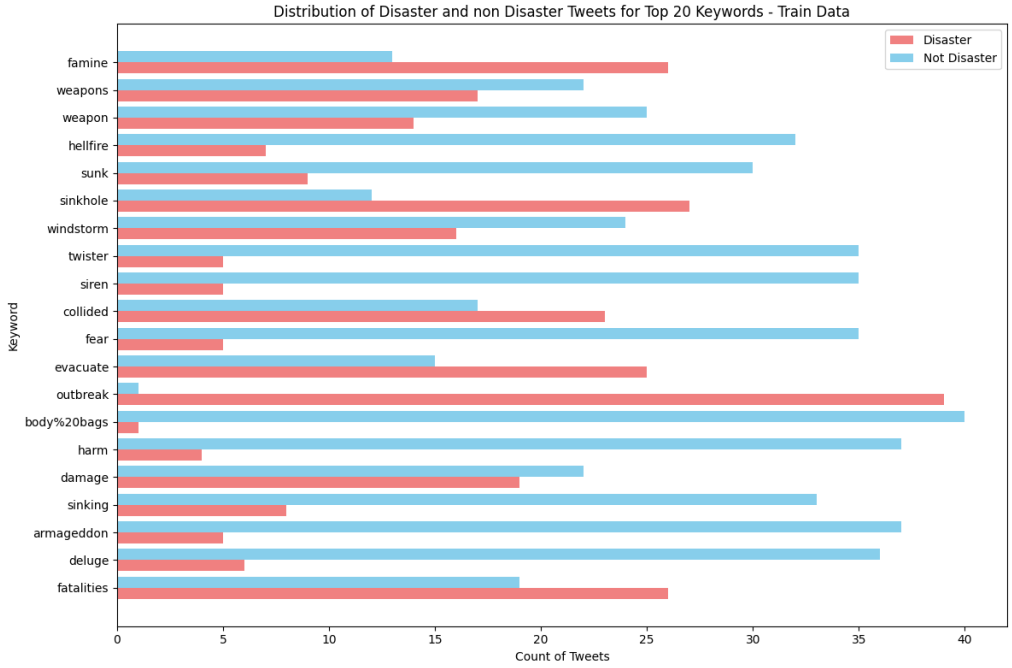

Figure 1: Distribution of disaster and non-disaster tweets for the top 20 keywords.

4. **Location Analysis and Distribution** A similar analysis was conducted for the location feature of the tweets. As shown in Figure 2, the location feature's top 20 locations with some exceptions shows significantly less skewness towards a certain label. This further indicated that the location feature might not be a valuable predictive feature to input to our models. ,

5. **Feature Engineering and Distribution Analysis** We engineered new features based on the tweets, including `text_length`, `punctuation_count`, and others. We also analyze the distribution of the target class with respect to these newly created features as shown in the Figure 3

   (a) `word_count`: The number of tokens in a tweet

   (b) `mention_count`: The number of mentions in the tweet

   (c) `hashtag_count`: The number of hashtags

   (d) `punctuation_count`: Number of characters from a set of punctuation marks and special characters.

   (e) `mean_word_length`: Average length of the tokens in the tweet

   While for `word_count` the class conditional distributions are mostly identical and overlapping, some other engineered features exhibit strongly differing distributions for the labels, especially in the case of `punctuation_count` or `mean_word_length`.These discriminative features could be leveraged as additional input to enhance the model's predictive capabilities.

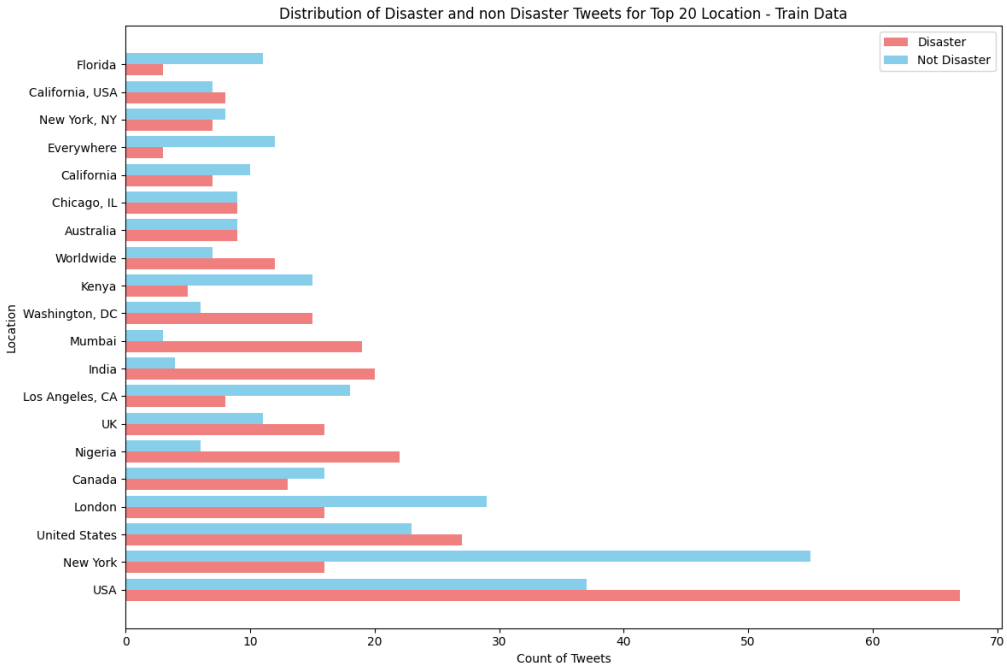

Figure 2: Distribution of disaster and non-disaster tweets for the top 20 locations.

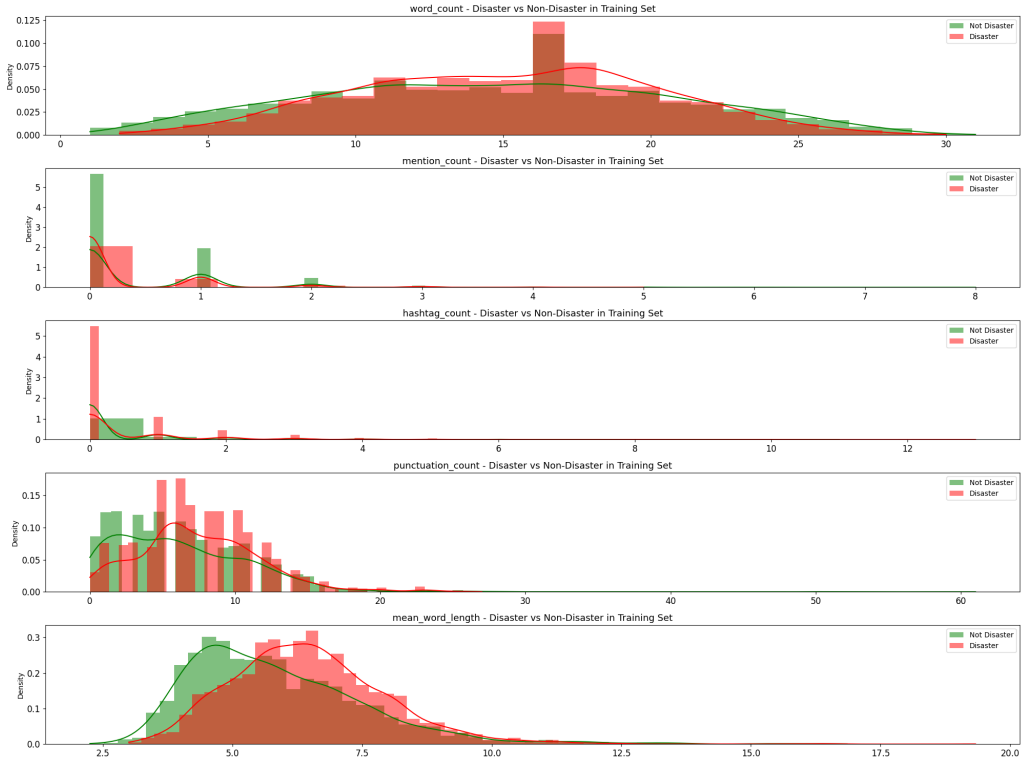

Figure 3: Distribution of disaster and non-disaster tweets for the engineered features

3 Data Processing

3.1 Data Cleaning

To better remove noise and inconsistencies in the messy tweet data, common data cleaning techniques were applied. The concrete steps taken were as follows:

1. Removing Number - Many usernames and miscellaneous words containing numbers were removed. normalization - Reduce the variation in text to a more standard word form by removing redundant characters and reducing repetition. For example, the word *"Hellllooo"* will normalize to *"Hello"*.

2. Splitting Hashtags - Hashtags such as #CaliforniaWildFire were split into their individual component, i.e. California Wildfire. This is done to allow embedding of the hashtags, which may convey important contextual information for classifying the tweet

3. Removal of Punctuation - removed punctuation and special characters

4. Removal of typical English stopwords - Words that have little semantic meaning were removed. The Natural Language Toolkit (NLTK) library was utilized for this filtering

Finally, the cleaned tweet was converted to lowercase and embedded into highdimensional word vectors

3.2 Data Embedding

A proper embedding that effectively captures the semantic relationship between terms is essential for ensuring good performance in the model. This is particularly important for tasks like tweet classification, where the text is often short, noisy, and lacks the structured context found in longer documents. Two popular word embeddings, Word2Vec and GloVe, were tested on our task.

3.2.1 Word2Vec

The word2vec embeddings were created by Google in 2013 and trained on 100 billion words from Google News. These embeddings consist of 300-dimensional vectors that are designed to capture semantic relationships between words. The model uses the Skip-Gram or Continuous Bag of Words (CBOW) approaches to predict either context words from a target word or the target word from context words. In our dataset, the word2vec embeddings had a 74.7% coverage of unique tokens. Since these embeddings were trained on Google News, it would not necessarily have embeddings for all of the words reflecting the ambiguous and highly-specific language that can be found in tweets.

3.2.2 GloVe Embeddings

We also experimented with the pre-trained word embeddings from the Global Vectors for Word Representation (GloVe) [1] project introduced by researchers from the Stanford Natural Language Processing group in the year 2014. It leverages global word-word-co-occurences information to capture the semantic relationships between words. For this classification task given the characteristic of tweets using a lot of slang and popculture terms, we opted for using pre-trained GloVe word vectors, that were trained on a large corpora of 2B twitter tweets, comprising of 27B words. This training data provides a deeper understanding of informal language and social media-specific words, crucial for the accurate classification.
Given the need for nuanced representation of word, we decided to use 100-dimensional word vectors. The pre-trained word embeddings covered a significant portion of our training dataset, encompassing approximately 92.86 % of the words. The words that fell outside the vocabulary of the embeddings were primarily proper nouns like names of natural disasters (e.g., "Haiyan," "Soudelor"), geographical locations (e.g., "Udhampur," "Nankana"), fictional characters (e.g., "Boruto") and other domain-specific terms. To handle words not found in the pre-trained GloVe vocabulary, we assigned random embeddings to them, allowing the model to learn representations for these unknown words during train. Additionally an `<UNK>` token was used to represent words that were entirely unseen during training. Due to the better coverage of the tokens in our tweets, we opted for using GloVe going forward.

4 Proposed Methods

4.1 LSTM

Why LSTMs? Let's first lay out the 3 reasons for which we chose an LSTM as the first model. 1. LSTMs can handle sequential data and store context, 2. LSTMs are good at dealing with ambiguity, 3. LSTMs are good at sentiment analysis.

The logic behind the first reason is that Tweets are written in natural language, which depends on the order and context in which words are used, so handling sequential data appropriately will allow the model to learn much more effectively. For example, looking at the simple Tweet "No earthquake reported today" compared to "Earthquake reported today", we hope that the LSTM would be able to pick up on the "no" in the second sentence, and maintain the context of negation until the final output.

Dealing with ambiguity is another crucial aspect, as tweets frequently use slang, sarcasm, or metaphorical language that would be hard for a regular model to identify. An LSTM's ability to learn contextual relationships helps it discern these nuances.

Finally, sentiment analysis is also highly relevant, as tweets about disasters often express urgency or distress, and detecting the sentiment behind the words can help determine whether a tweet is related to a real emergency or not. An LSTM can effectively capture these sentiment cues within the sequence of words to make more accurate classifications.

4.2 LSTM with GloVe

4.2.1 Model architecture

The baseline LSTm model begins with the pre-trained GloVe embedding layer. The word embeddings resulting from the cleaned tweets will then be input to two consecutive LSTM layers: the first with 64 and the second with 32 units. Both layers incorporate a recurrent dropout rate of 0.01 and a regular dropout rate of 0.1 for improved generalization. Following the LSTMs, there is a dense layer with 32 units, ReLU activation, and L2 regularization to capture high-level features. The final output layer leverages a sigmoid activation to produce probabilities.

4.2.2 Model performance

With this baseline architecture an accuracy of 82.01% on the validation set and 80.69% on the test dataset of the competition was achieved. The validation set was created by randomly splitting 20% of the training data.

4.3 LSTM with Word2Vec

With the Word2Vec embeddings we tested the same model architectures and managed to reach a validation accuracy of 80.16% and a test accuracy of 77.43% before modification, and a test accuracy of 80.12% after modification (mentioned below).

4.4 Multi-Input LSTM modification

In an attempt to further improve the performance of the model, more predictive features such as `punctuation_count`, `mean_word_length` and the keyword of the tweet were incorporated:

4.4.1 Model architecture

The preproccessed tweets still is passed through 2 LSTM layers with 64 units and 32 units respectively. Additionally now, a separate learnable embedding layer processed the keyword inputs and additional dense layers with ReLU activation processed the punctuation_count and mean_word_length of each tweet. Finally the outputs were concatenated, followed by a dense layer (16 units, ReLU activation) to learn complex relationships of the diverse inputs and finally the output layer generated a binary classification.

4.4.2 Performance and Thoughts

To our surprise the added inputs did not improve the model's performance. The validation accuracy dropped to 81. 65% and the overall accuracy in the test data set decreased to 80. 44%. This suggests potential overfitting and negative feature impact
The increased model complexity introduced by these extra features may have led to overfitting on the training data, hindering its generalization ability. Additionally, the added features, particularly the keyword feature, might have introduced noise or inconsistencies, negatively affecting the model's performance It is crucial to acknowledge that the analysis of keyword bias was limited to the top 20 most common keywords. This restricted view might not accurately represent the overall behavior of keywords across all labels, especially when considering the test data.

4.5 MLP + *Spacy*

After implementing the LSTM model with GloVe embeddings, we gained a deeper understanding of the effectiveness of text embeddings in capturing semantic information from textual data. In the introduction, we had framed this task as a binary classification problem, where the objective was to assign one of two possible classes (Disaster, Not a Disaster) to the given inputs. While the LSTM model demonstrated the power of using embeddings for text-based predictions, we decided to explore other approaches to provide a broader perspective on solving this problem. Specifically, we turned to more traditional classification algorithms, which are known for their simplicity and effectiveness in various machine learning tasks. Classical algorithms such as Support Vector Machines and Logistic Regression are often employed in similar scenarios, offering robust performance for classification problems. With this in mind, we opted to train a Multi-Layer Perceptron model, leveraging the text embeddings as input features. This approach allowed us to combine the representational power of embeddings with the flexibility of a neural network-based classifier.

4.5.1 MLP Architecture

We quickly implemented a MLP with one hidden layer using *PyTorch*. The first hidden layer consists of 64 neurons, followed by another hidden layer with 32 neurons. For the input, we used a *spaCy*-generated text embedding of size 300. To improve generalization and prevent over-fitting, we incorporated a dropout layer after the first hidden layer. Each hidden layer uses the *ReLU* activation function to introduce non-linearity and enable the model to learn complex patterns in the data. After the network, the output passes through a *sigmoid* activation function, which scales the results to a range between 0 and 1. This output format makes it suitable for training with the Binary Cross-Entropy loss function, commonly used for binary classification tasks. The final prediction is then obtained by comparing the neural network result to a threshold (0.5 for the final model).

4.5.2 *Spacy* Model

For this approach, we chose not to use the *GloVe* embeddings. Instead, in the spirit of experimentation and comparison, we utilized the *spaCy* model "*en_core_web_lg*" [2] . This model is trained on a diverse range of internet data, including blogs, news articles, and comments, making it well-suited (via it's **tok2vec** layer) for generating high-quality text embeddings for twitter data.

4.5.3 Results and Other thoughts

With our architecture and embedding generator in place, the next step was to experiment with the available data to maximize accuracy. For instance, we explored incorporating additional features such as tweet keywords and location, either by concatenating them with the text or using them as multi-inputs to the network. However, these approaches resulted in lower accuracy compared to a simpler network that used only the tweet text embeddings as input. Using this MLP architecture and the selected embeddings, we achieved an accuracy of 82.51% on the validation dataset and 80.90% on the test dataset. The training process was notably fast, but we observed that the model started overfitting after just 10 epochs, indicating the need for regularization or a lack of data.

4.6 DistilBERT

4.6.1 Overview of the Model

For further experiments, we utilized a LLM model proposed in 2019, called Distilled BERT [3]. DistilBERT is a lightweight version of the original BERT model, introduced in 2018. BERT, trained on Wikipedia and BookCorpus datasets, significantly outperformed previous SOTA models, and is now considered a baseline model for many NLP tasks. DistilBERT, on the other hand, is a 66M parameters encoder-only language model, which is 40% smaller than the original BERT model, however, maintains 97% of its performance. Its general structure is presented in Figure 4.

Since the base model of distilBERT outputs a layer comprising of contextual word embeddings, for majority of tasks, it can be extended with a particular *head model*. For instance, for sequence classification tasks, we can simply prepend a MLP to the last layer, with a softmax output layer, and train only the classification head. In addition to that, frameworks such as HugginFace[1] provides not only the pre-trained model, but also the same model with a prepended classification head, which we then can utilize for our task.

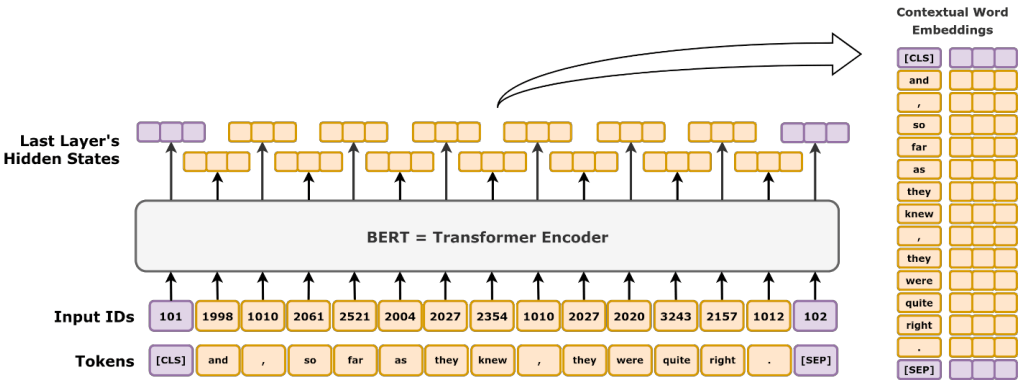

Figure 4: General Architecture of BERT Model

4.6.2 Proposed Methods

For our particular task, we present three main approaches that we used to train the distilBERT model. We ensured that the models were trained on the same data and performed experimets several times to account for randomness.

Approach I: Classification Head Training. The simplest way of training the model. We froze all the pre-trained NLP layers, and only trained the classification head, which has around 500k parameters. We did that several times to not be dependent on initialization, testing each model on validation data. The best final result that we got on validation set was 82.3%, and for Kaggle test set was **81.00**%.

Approach II: Whole Model Training We then theorized that modifying the weights of the base model may improve the performance, as it could make the resulting contextual embeddings better fit the tweets' text distribution. Therefore, we unfroze almost all the layers and trained the model. The final best accuracy on validation set was 82.33%, however, on Kaggle, the accuracy actually even slightly lower than for approach 1: **80.91%**.

Approach III: Finding Optimal Layers to Train Finally, we combined both approaches. Since the performance was still not as high compared to other teams' submissions, we theorized that freezing only certain components of the base model and the resulting classification head from Approach II, may be beneficial. However, the number of combinations of which components (given by the implementation of the model by *transformers* library) to freeze is equal to $n = 2^{102}$, which is a very high number. However, in order to find the optimal combination, for each one of them, we would need to train a new model and evaluate its performance!

[1] `huggingface.co/`

This is obviously infeasible given the computational power we have, therefore, we proposed to approximate the solution by using a powerful, but simple approach: Genetic Algorithm [4]. The algorithm we propose is the following:

1. A population of random chromosomes is initialized, where each chromosome represents which layers in the model are frozen, and which are trainable. For instance, a chromosome $[1, 0, 1]$, represents a model, whose first and third components are be trainable, and the second one is frozen.
2. For each chromosome, we construct a new model, where the original model's components are labeled to either trainable or not, depending on the corresponding gene in the chromosome.
3. The fitness of each chromosome is computed. The fitness function in our case is simply the validation loss.
4. The fittest chromosomes of the population are extracted, and the next generation is constructed using the fittest chromosomes. This is done by combining several GA Operations:
   - Elitism. Top chromosomes are passed to the next generation without any modifications.
   - Crossover. The sub-sequences of two fit chromosomes are combined to compute a possibly superior child.
   - Mutation. With a certain probability, genes in a chromosome are mutated to different bit values.
   - Mutated elitism. More harshly mutated **elite** chromosomes are passed to the next generation.
5. Steps [2 - 5] are taken for a specific number of times or until convergence.

We performed the GA several times with different initial populations and seeds, and the best result that we got did not significantly improve and corresponded to the final validation accuracy of test accuracy of 82.40, and Kaggle's test accuracy of **80.94**%. Note that this did not improve compared to the first appraoch.

Based on these results, we can very clearly conclude that choosing to retrain the NLP layers in order to improve the performance of the classification head does not gradually improve the performance!

4.6.3 Model's Quantization

Lastly, we decided to experiment with another approach that is not related to obtaining optimal accuracy in the Kaggle competition. Since Distil-BERT is the most complex model of the three porposed approaches, however, obtains similar results to the initial models that have significantly less parameters, we tried to find a way to reduce the size of the model without losing performance. This aim to reduce of size of the model is called Neural Network Quantization.

Overview of Existing Quantization Methods Quantization of large language models is a well-established and evolving area of machine learning research. Numerous quantization algorithms are already used to optimize models for various cases[2]. However, research in quantization remains highly active with new, more efficient techniques being developed (for instance, [5]). Generally, quantization algorithms are divided in three categories:

- **Post Training Dynamic Quantization (PTDQ).** The range (precision) of each activation is computed during the runtime. This approach provides positive results without much preprocessing effort, as it eliminates the need for a calibration dataset and does not involve model retraining or fine-tuning.
- **Post Training Static Quantization (PTSQ).** The range for each activation is computed in advance at *quantization-time*, usually by passing representative data (calibration dataset) through the model and recording the activation values.
- **Quantization Aware Training (QAT)**. The range for each activation is computed at training-time: the loss of the training also depends on the degree of quantization.

Currently, there exist several state-of-the-art PTQ algorithms used for LLM quantization.

[2] `https://huggingface.co/docs/transformers/v4.46.0/quantization/overview`

- **GPTQ** ([6]). The working principle is based on minimizing the quantization error using second-order optimization. GPTQ is extremely efficient: it can quantize GPT models with 175 billion parameters in approximately four GPU hours while maintaining reasonable accuracy
- **AQLM**([5]). Achieves extreme compression by breaking down weight matrices into smaller components and optimizing them in a way that adapts to the input structure of the model. It achieves significant results when encoding weights using just 2-3 bits without sacrificing much performance.

However, the majority of the algorithms are already implemented by Huggin Face framework, and therefore, to implement them on our model, we would only need to write several lines of code. Thus, in this report, we propose a new simple method to quantize the NN models, and present the results.

Proposed GA-Based Quantization Approach The method we propose is very similar to the one proposed in 4.5.2. Approach III, as we once again utilize the GA. The only key difference is that now a gene in chromosome depicts the amount of bits that layer is compressed to. The main idea of the algorithm is that we aim to find the optimal combination of layers to quantize to lower bit values, as we theorize that there exists a combination of layers that, even when quantized using a very simplistic approach, would result in non-worse performance compared the original model.

The method we propose is the following.

1. A population of random chromosomes is initialized, where each chromosome represents a possible version of quantized NN. For instance, a chromosome $[2, 1, 32]$, represents a model, whose first layer is quantized to int2, second layer to int1 and the third to float32.
2. For each chromosome, the original NN is transformed to its quantized version (for that we simple use Uniform quantization[3], and its performance on the validation data is computed.
3. The fitness of each chromosome is computed. The fitness function outputs a score of the overall effectiveness of a given chromosome in balancing validation loss and quantization efficiency. In particular, the fitness function used in our algorithm is the following (note that we consider classification tasks!):

   $$f(\mathbf{X}) = \lambda_1 l_{\text{validation loss}}(\mathbf{X}) + \lambda_2 q_{\text{quantization score}} + \lambda_3(\max(0, c_{\text{min accuracy}} - f_{\text{validation mean accuracy}}(\mathbf{X})),$$

   where $\mathbf{X}$ represents the list of all quantized weight matrices, and quantization score represents the normalized amount of bits it takes to encode all weights:

   $$l_{\text{quantization score}} = \frac{1}{n_{layers} \cdot n_{\text{weights}}} \sum^{n_{\text{layers}}} c_i x_i,$$

   where x_i is the amount of weights in layer i, and the c_i represent the i^{th} gene in the chromosome.

   Note that in the given fitness function, by altering the values of $\lambda_1, \lambda_2, \lambda_3$ we can control how important the compression rate of the model is, compared to maintaining the performance on the validation data. In addition, when a high λ_3 is chosen, the third term allows us to set a minimum value to which the model's accuracy may drop!
4. The fittest chromosomes of the population are extracted, out of which the next generation is constructed. This is done the same way as in 4.5.2 Approach III, step 5.

   We have utilized this algorithm on the best-performing model with different GA parameters. We set the parameters in a way that the final accuracy of the quantized model is almost no less than the one obtained by the best-performing model, namely 82.40,%. Initially, it would take 2142560320 bits to encode the model, however, after applying our algorithm and fine-tuning parameters, we managed to theoretically compress it to 535658512 bits, which corresponds to almost **75**% of size compression!

[3] `huggingface.co/docs/optimum/concept_guides/quantization`

5 Final Results

The final overview of the obtained results is presented in Table 1. We can clearly see that all approaches resulted in rather similar results, while Distil-BERT performed slightly better than the other two. The reason for that very likely lies within model's ability to slightly better generalize patterns, as it is a way bigger model compared to the rest of presented approaches. Lastly, we managed to show that the model could be compressed to 25% of its size and still achieve the same accuracy.

Approach	Test Accuracy
LSTM+Glove	80.69 %
Multi-Input LSTM	80.44 %
MLP+Spacy	80.90%
Distil-BERT	81.00%

Table 1: Overview of results

6 Summary

To conclude, we started this on this project with an evident goal: solving a simple binary classification problem. However, it appears that it was more than that and and it served as an invaluable learning experience. Understanding and processing data is one of the most crucial aspect of big data intelligence; so we began by analyzing the data and cleaning it up using some traditional NLP techniques. After that we tried to solve the problem with 3 different approaches, increasing gradually our model's accuracy and gaining knowledge.

Future Research Directions There are a lot of directions for future research on this task. One key area is finding an effective way to incorporate auxiliary data (such as keywords, locations, and text length) into the model without introducing noise, thereby enhancing its classification capabilities. Additionally, given the limited amount of data available, we also thought it could be interesting to use generative methods like Variational Auto-encoders to create more training samples and see how it goes. This raises questions such as: Would the model overfit? Or could random sampling improve its generalization capabilities? Another important direction involves Out-of-Distribution (OOD) data detection to evaluate how much we can trust the model when it comes to unseen data. Such research could significantly improve the reliability of the model and would be a another source of knowledge for us.

Potential Real-world Applications Since this work was conducted on real-world data, it opens the door to practical applications. For instance, we envisioned creating a Twitter bot that could classify tweets as potential disaster-related or not. This could be implemented via community notes or automated tweet replies. Additionally, a real-time monitoring website could be developed, continuously gathering and displaying relevant information for users.

Finally, thank you for reading our report! We hope you found it insightful and look forward to your feedback and suggestions.

References

[1] J. Pennington, R. Socher, and C. D. Manning, *GloVe: Global Vectors for Word Representation*, `https://nlp.stanford.edu/projects/glove/`, [Online].

[2] Explosion, *Spacy en_core_web_lg*, `https://spacy.io/models/en#en_core_web_lg`, [Online].

[3] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, *Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter*, 2020. arXiv: `1910.01108 [cs.CL]`.

[4] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992, ISSN: 00368733, 19467087.

[5] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, and D. Alistarh, *Extreme compression of large language models via additive quantization*, 2024. arXiv: 2401.06118 [cs.LG].

[6] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, *Gptq: Accurate post-training quantization for generative pre-trained transformers*, 2023. arXiv: 2210.17323 [cs.LG].