Emotion detection through text data using a 3-mechanism hybrid model (Transformer Neural Networks, Fuzzy Systems & Evolutionary Algorithms)

Georgios Lekkas
gxlekkas@gmail.com
gelekka@cs.ihu.gr

Introduction

For the second part of our emotion detection project we have developed and implemented a 3-mechanism hybrid model using the BERT Transformer Neural Network designed for Natural Language Processing (NLP), Fuzzy Logic Systems and Genetic Algorithms (GAs) a variant of Evolutionary Algorithms (EAs). In order to classify text and analyze sentiment, we have made use of the publicly available IMDB Large Movie Review v1.0 (http://ai.stanford.edu/~amaas/data/sentiment/).

Data preparation and preprocessing

The IMDB dataset contains 50.000 movie reviews splitted evenly into 25.000 reviews as train and 25.000 reviews as test data. Each 25.000 dataset contains 12.500 positive reviews and 12.500 negative reviews. In the labeled train and test set, a negative review has a score <= 4 out of 10, and a positive review has a score >= 7 out of 10. Thus reviews with more neutral ratings are not included in the train/test sets. In the unsupervised set, reviews of any rating are included and there are an even number of reviews > 5 and <= 5. Baring in mind the available time to complete this project, we have used only the supervised data set, thus conducting only supervised machine learning. The two sets (train and test) contain [pos, neg] directories for the reviews with binary labels positive and negative. Within these directories, reviews are stored in text files named following the convention [[id]_[rating].txt] where [id] is a unique id and [rating] is the star rating for that review on a 1-10 scale. For example, the file [test/pos/200_8.txt] is the text for a positive labeled test set example with unique id 200 and star rating 8/10 from IMDb. The [train/unsup/] directory has 0 for all ratings because the ratings are omitted for this portion of the dataset in order to be used with an unsupervised approach.

Initially, we have developed two scripts to load and preprocess the data (Load_data.py & Data_preprocessing.py). From both the test and the train data, text files containing positive and negative sentiment reviews are loaded from specified directories, with each review's sentiment and rating extracted and compiled into a DataFrame. This raw data undergoes preprocessing to clean and standardize the text, which involves removing HTML tags, URLs, numbers, and punctuation, converting text to lowercase, tokenizing, removing stopwords, and lemmatizing. The cleaned texts are then vectorized using TF-IDF (Term Frequency-Inverse Document Frequency), transforming them into a format suitable for machine learning model input. After the preprocessing, the text are ready to get fed into the BERT transformer model. Lastly, to confirm the existence of a evenly distributed data-set, we performed an Explanatory Data Analysis (EDA.py) producing a class distribution (sentiment the count) (Figure 1), confirming the existing of 12.500 reviews per sentiment (pos / neg) and a word-cloud (Figure 2) – a visual representation of a text (bigger words appear more often) of the processed data.

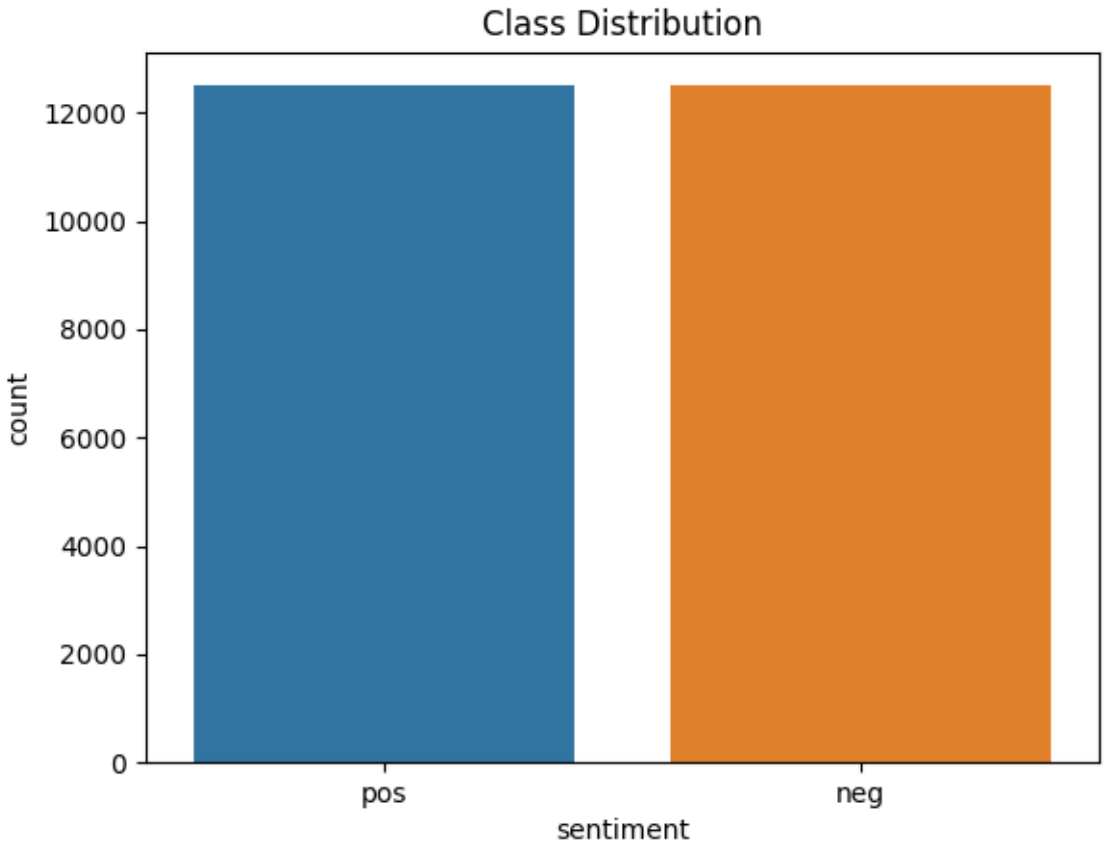

Figure 1: Class distribution (sentiment vs count)

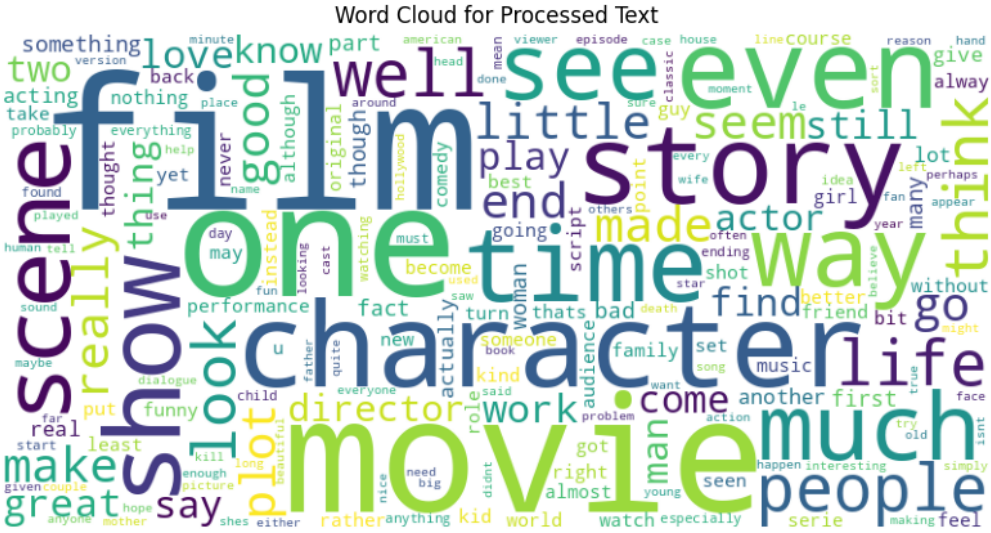

Figure 2: Word-cloud

BERT Model

Training / Fine-Tuning & Performance Evaluation

Proceeding to implement our first model, we have developed our scripts to train, fine-tune, and evaluate the performance of our fine-tuned model. (Implement BERT Transformer.py, Evaluate BERT performace metrics.py). The process begins by leveraging a pre-trained BERT model for sentiment analysis by fine-tuning it on the train dataset. Despite the fact that the model assesses the availability of GPUs for computing via the CUDA function (Nvidia GPUs), our system lacks the availability of a GPU so the whole script ran only on CPU which costs us time. More will be discussed on the discussion section.

The dataset, already processed and labeled for sentiment, is loaded and prepared for the BERT model; this involves tokenizing the texts, by splitting them into tokens that BERT can understand, adding special tokens, generating attention masks for handling varying sequence lengths, and converting these elements into tensors. The core of the script involves setting up training and validation loops. A BERT model for sequence classification is initialized, along with an optimizer and a learning rate scheduler to manage the learning process efficiently. During training, the model learns from the data in batches, updating its weights to minimize the classification loss. This process is carefully regulated to prevent overfitting and ensure gradient stability. Following each epoch, the model's performance is evaluated on a separate validation set to monitor its ability to generalize to unseen data. Upon completing the training epochs, the model, now fine-tuned to the specific task of sentiment analysis, along with its tokenizer, is saved for future use.

Then, the evaluation of the performance of the BERT Model takes place. The test dataset now is loaded, which contains preprocessed text data alongside their corresponding sentiment labels ('pos' for positive and 'neg' for negative), converting these labels into a numerical format for processing. In the evaluation loop, the script disables gradient calculations to save memory and computation since back-propagation is not needed for prediction. For each batch of test data, it feeds the input IDs and attention masks to the model, receiving logits (raw model outputs before applying an activation function) in return. These logits are processed to predict sentiment labels, which are then compared against the true labels to calculate accuracy and other performance metrics such as precision, recall, and F1-score. Finally, the script consolidates these performance metrics, including a confusion matrix that visually represents the model's predictions in relation to the true sentiments, and prints them. The performance metrics are illustrated on table 1.

Performance Metrics	Precision	Recall	F1-Score	Support
0	0.90	0.91	0.91	12500
1	0.91	0.90	0.90	12500
Accuracy			**0.90488**	25000
Macro avg	0.90	0.90	0.90	25000
Weighted avg	0.90	0.90	0.90	25000
Confusion Matrix	TN = 11343	FP = 1157		
	FN = 1221	TP = 11279		

- **Accuracy**: **0.90488.** This indicates that the model correctly predicted the sentiment of approximately 90.5% of the test dataset.
- **Precision** for class 0 (negative sentiment) is 0.90, meaning that 90% of instances predicted as negative are indeed negative. For class 1 (positive sentiment), precision is slightly higher at 0.91, indicating that 91% of instances predicted as positive are correctly identified.
- **Recall:** For class 0 is 0.91, showing that the model successfully identified 91% of all actual negative instances. For class 1, recall is slightly lower at 0.90, meaning 90% of actual positive instances were correctly predicted.
- **F1-score:** Provides a balance between precision and recall. For both classes, the scores are close (0.91 for negative and 0.90 for positive), suggesting a well-balanced model performance between precision and recall.
- **Support:** Indicates the total number of instances for each class in the test data, here equally divided with 12,500 instances per class.

Confusion Matrix:

- **True Negatives (TN): 11,343** - The number of negative instances correctly identified.
- **False Positives (FP): 1,157** - The number of negative instances incorrectly labeled as positive.
- **False Negatives (FN): 1,221** - The number of positive instances incorrectly labeled as negative.
- **True Positives (TP): 11,279** - The number of positive instances correctly identified.

Fuzzy Logic System

Proceeding with the methodology, now we are ready to implement our fuzzy logic system and integrate it with the fine-tuned BERT model building a hybrid approach (BERT_and_Fuzzy_System_Implementation.py). The process begins by loading a pretrained BERT model and tokenizer, specifically fine-tuned for sequence classification tasks. This model is adept at understanding and analyzing the sentiment of text data, providing a sentiment score based on the textual input it receives.

The core of this hybrid system lies in the creation of a fuzzy logic control system that interprets the sentiment score output by the BERT model. The fuzzy logic part of the system introduces a level of interpretability and flexibility that is especially useful in handling the ambiguity inherent in natural language. Fuzzy logic, unlike binary logic, allows for reasoning about data that can belong to multiple classes or categories simultaneously, to varying degrees. This is particularly beneficial for sentiment analysis, where sentiments can often be a blend of positive, negative, and neutral feelings.

Two linguistic **variables** are defined: **sentiment** and **output**. The sentiment variable represents the input, which is the sentiment score derived from the BERT model, and the output variable represents the sentiment classification result (positive or negative). These variables are defined over specific ranges, allowing for the representation of sentiment intensity on a continuum from 0 to 1.

Membership functions are also defined: For each linguistic variable, membership functions define how each point in the variable's universe (the range of values it can take) corresponds to a degree of membership in a certain category. In the script:

- **Sentiment**: This variable has membership functions for 'negative' and 'positive', essentially categorizing the BERT model's sentiment scores into these two broad sentiments.
- **Output**: This variable also has membership functions for 'negative' and 'positive', representing the fuzzy system's sentiment classification output.

These functions are designed to capture the fuzziness of sentiment classification, where a sentiment score might not strictly belong to one category but can have varying degrees of membership in both.

Fuzzy Rules: The core of the fuzzy system's decision-making process lies in its rules. In this script, rules are defined to classify the overall sentiment based on the input sentiment score. For example:

- If the sentiment score is more aligned with the 'negative' membership function – BERT model's sentiment score falls within the range defined for 'negative' sentiment the output is classified as 'negative'.
- Conversely, if the sentiment score aligns more with the 'positive' membership function – BERT model's sentiment score falls within the range defined for 'positive' sentiment the output is classified as 'positive'.

These rules leverage the fuzzy membership functions to make nuanced decisions. They allow for cases where the sentiment score might be somewhat positive but not entirely, reflecting the complexity and nuance of human emotions in textual data.

Upon receiving textual input, the script tokenizes and preprocesses the text using the BERT tokenizer, then feeds it into the BERT model to obtain a raw sentiment score. This score is then interpreted by the fuzzy logic system, which classifies the sentiment as either 'positive' or 'negative' based on the defined rules and membership functions.

An example usage of the script demonstrates how it processes a detailed and emotionally rich text, extracts a sentiment score using the BERT model, and then classifies the sentiment through fuzzy logic. The result is a nuanced sentiment analysis that leverages the strengths of both BERT's deep learning capabilities and fuzzy logic's interpretability to provide a sophisticated understanding of text sentiment. An example is presented in Figures 3 and 4. Figure 3 illustrates the outcome of the IDE console, whereas Figure 4 presents the outcome of a mini interface constructed for a more beautiful and simple use (Executable_script.py).

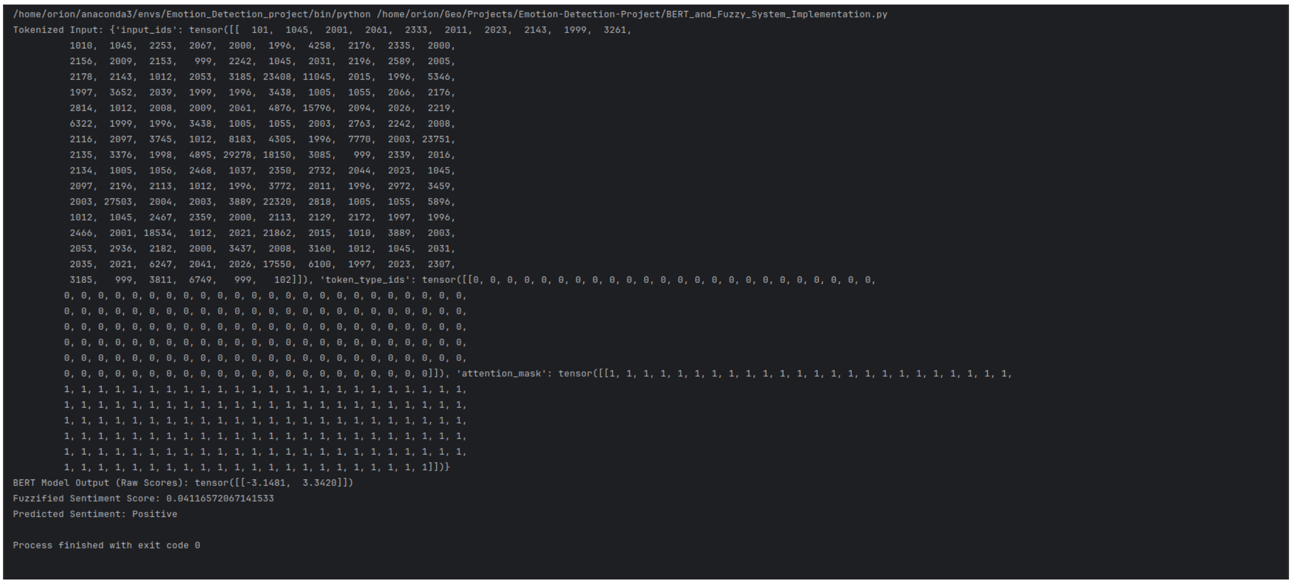

Figure 3: The output of the console

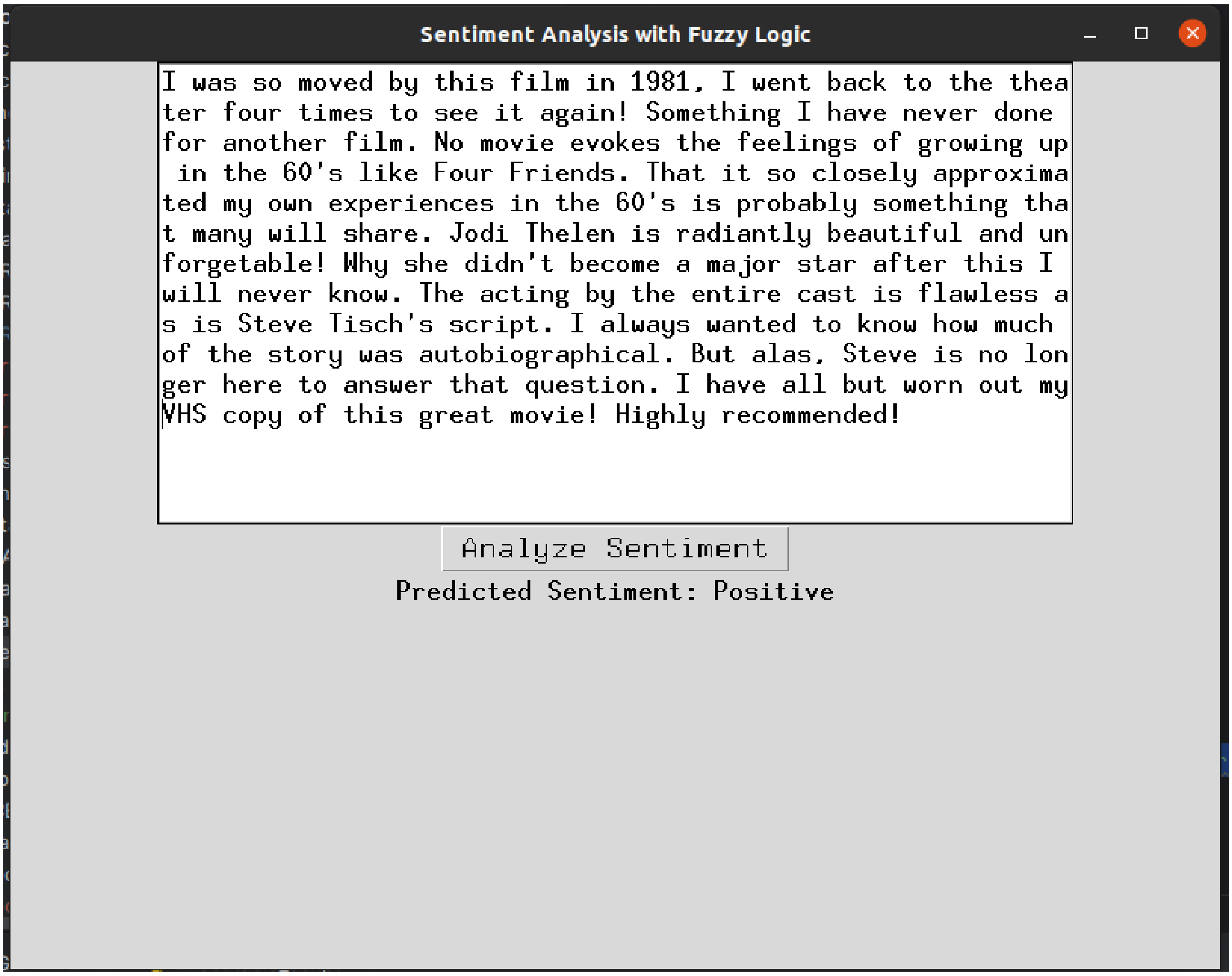

Figure 4: Interface output

Genetic Algorithms (GAs)

To complete our 3 – mechanism hybrid emotion detection project, we implement a Genetic Algorithm, a variant of the Evolutionary Algorithms (Genetic_Algorithm.py). The system begins by loading a fine-tuned BERT model and tokenizer, which are used to process textual input and extract sentiment scores. These sentiment scores serve as inputs to a fuzzy logic system that classifies the sentiment as positive or negative based on rules defined within the system, as we discussed before.

The fuzzy logic system is dynamically adjusted using parameters optimized by a genetic algorithm. This GA operates by evolving a population of individuals, where each individual represents a set of parameters that define the fuzzy system's membership functions for classifying sentiments. These parameters are initially set to random values and iteratively adjusted through genetic operations such as selection, crossover, and mutation to maximize the system's performance on a given dataset. The performance is evaluated based on accuracy, precision, recall, and F1 score metrics, calculated by comparing the fuzzy system's predictions against the actual sentiments of a test dataset.

The GA's objective is to find the optimal set of parameters that result in the highest accuracy and overall performance of the sentiment classification system. Once the optimal parameters are identified, the script prints these parameters along with the system's performance metrics. The system's efficacy is demonstrated through its ability to adapt and fine-tune the fuzzy logic rules and membership functions based on feedback from the genetic algorithm. Lets dive deeper on the separate components.

Genetic Algorithms operate on a population of potential solutions, evolving these solutions over time through a process akin to biological evolution. This involves selection based on fitness, crossover (or recombination) to produce new offspring, and mutation to introduce variations. GAs are particularly useful for problems where the search space is vast and complex, making traditional optimization methods impractical.

In the our script, the GA is implemented with the help of the DEAP library, a comprehensive toolkit for evolutionary computations. Key components of the GA in the script include:

- **Individual Representation**: Each individual in the population represents a set of parameters for the fuzzy logic system. These parameters are encoded as lists of floating-point numbers, which might define the shapes and positions of the membership functions used within the fuzzy system. This encoding allows the GA to adjust the fuzzy logic system's behavior by evolving these parameters.
- **Fitness Function**: The core of the GA's optimization process is the evaluation of each individual's fitness, which is a measure of how well an individual solves the given problem. The ‘evaluate’ function in the script calculates the performance metrics (accuracy, precision, recall, F1 score) of the fuzzy system configured by an individual's parameters. These metrics are obtained by comparing the fuzzy system's sentiment predictions against a set of test data. The goal is to maximize these metrics, with higher values indicating better solutions.
- **Genetic Operators**:

  The script employs several genetic operators to evolve the population:

    - **Selection**: This operator selects individuals for reproduction based on their fitness, preferring those with higher fitness values. It simulates the natural selection process, where fitter individuals are more likely to pass on their genes.
    - **Crossover**: Crossover combines pairs of individuals (parents) to produce offspring for the next generation. It mimics sexual reproduction, allowing offspring to inherit characteristics from both parents, potentially leading to better solutions.
    - **Mutation**: Mutation introduces random changes to some individuals' traits. This process ensures diversity within the population, preventing premature convergence to sub-optimal solutions and exploring new areas of the search space.
- **Evolution Process**: The 'main' function orchestrates the evolution of the population through a specified number of generations. During each generation, the population undergoes selection, crossover, and mutation. The fittest individuals are identified, and their fitness scores are used to guide the evolutionary process towards increasingly better solutions.

A CSV file with test data is provided at the GA. The CSV file comprising text entries along with their corresponding sentiment labels (positive or negative). This data-set is essential for assessing how well a given configuration of the fuzzy logic system performs in terms of sentiment analysis. By comparing the sentiment predictions made by the fuzzy logic system against the actual sentiments in the CSV file, the GA can calculate performance metrics such as accuracy, precision, recall, and F1 score. These metrics provide a quantitative basis for evaluating the fitness of each configuration. The performance metrics derived from the test dataset serve as feedback to the GA, guiding the evolutionary process. The GA aims to maximize these metrics by evolving the parameters of the fuzzy logic system across generations. Thus, the CSV file indirectly influences the direction and efficiency of the optimization, leading to progressively better configurations of the fuzzy logic system. The CSV file allows for consistent bench-marking across different runs of the GA. By using the same data-set for evaluation, one can fairly compare the performance of various configurations and iterations of the fuzzy logic system, ensuring that improvements or regressions in performance are accurately attributed to changes in the system's parameters rather than variations in the test data. In summary, the CSV file plays a pivotal role in the GA-based optimization process for the fuzzy logic system, providing the necessary data for evaluating, bench-marking, and guiding the evolution of system configurations towards optimal performance in sentiment analysis tasks.

Discussion

In this project, we've developed a sophisticated three-mechanism model for emotion detection and sentiment analysis that integrates the strengths of BERT, fuzzy logic, and genetic algorithms. At its core, the model utilizes a fine-tuned BERT (Bidirectional Encoder Representations from Transformers) model, leveraging its advanced natural language processing capabilities to understand and classify textual sentiment accurately. This deep learning model has been specifically fine-tuned on sentiment analysis data, making it adept at capturing the nuances of emotional expression in text.

To enhance the interpretability and handling of nuanced or ambiguous sentiments, the model incorporates a fuzzy logic system. This system uses predefined rules and membership functions to process the sentiment scores from BERT, enabling it to deal with the inherent vagueness and subjectivity of human emotions. The fuzzy logic component provides a layer of

interpretive logic that can classify sentiments with greater granularity, offering insights that are more aligned with human perception.

The integration of these two systems is further optimized by genetic algorithms (GAs), which dynamically adjust the parameters of the fuzzy logic system to maximize performance. The GA explores various configurations, selecting the most effective ones based on their ability to accurately classify sentiments. This optimization process is crucial for tailoring the model to specific data-sets and sentiment analysis challenges, ensuring that the system remains robust across diverse textual contexts.

The main obstacle which greatly affected the implementation of our model is the computational power of our local system. Both the fine-tuning and evaluation process needed 2 days each in order to complete because our local machine lacks the availability of a NVIDIA GPU to be used by the CUDA function which enables GPU acceleration and computation. So to run all the scripts, we had to use a quad core AMD CPU for all our computations. Upon at the time of this writing, the last model, the Genetic Algorithm (Genetic_Algorithm.py), is still running. The output of the algorithm, called as “Best Parameters”, represent the optimized values for the membership functions of the fuzzy system. These parameters will be used to maximize the accuracy of the sentiment analysis. In this way we will find out the best limits for the ‘negative’ and ‘positive’ membership functions, how the fuzzy logic system defines if an input sentence has a positive or negative sentiment, so we will find out the optimal configuration of the fuzzy system.

Conclusion

In conclusion, the three-mechanism model combining BERT, fuzzy logic, and genetic algorithms represents a solid advancement in sentiment analysis, offering nuanced interpretations of text sentiment and emotion. By leveraging the strengths of each component, this hybrid model can achieve high accuracy and adaptability, making it a valuable tool for understanding complex emotional expressions in textual data.

Some thoughts about a future work on this endeavor, the model presents a solid foundation for extending beyond binary sentiment classification to detect a broader spectrum of emotions, such as happiness, anger, enthusiasm and more. This expansion could involve refining the fuzzy logic system to include membership functions and rules specific to these emotions, enabling the model to distinguish between them with greater precision. Additionally, incorporating multi-label classification techniques within the BERT framework could allow for the detection of multiple emotions within a single text, reflecting the rich and layered nature of human emotional expression. Further optimization with genetic algorithms would continue to enhance the model's performance, tailoring it to specific data-sets and emotional nuances. Expanding the model in these directions would not only increase its utility across various applications but also deepen our understanding of emotional dynamics in text-based communication.

All the developed scripts can be found at my GitHub directory: https://github.com/GeoLek/Emotion-Detection-Project