Sentiment Analysis on Reddit Comments and Posts for OSINT Purposes

Rahul Ravi Hulli
M.S. Information Systems Security (CIISE)
Concordia University
Montreal, Quebec, Canada
rahulravi.hulli@gmail.com

***Abstract*—Abstract—The goal of this project is to evaluate articles and comments on Reddit for OSINT Purposes. One of the biggest conversation sites on the internet, Reddit offers a plethora of content created by its users, which makes it a wonderful place to learn about attitudes, communities, and trends in the online community. The main goals are to predict posts and use sentiment analysis to gauge the depression. The goal is to advance the field of social media analytics and deepen the knowledge of online speech and behavior.**

***Index Terms*—Reddit, Sentiment Analysis, Machine Learning, LSTM, GRU, Custom Algorithms, Depression**

I. Introduction

Reddit's policies on violent content aim to maintain a balance between fostering free expression and ensuring a safe and respectful environment for its users [1]. The platform explicitly prohibits the posting of content that promotes or glorifies violence against individuals or groups, including but not limited to credible threats, terrorist propaganda, graphic violence lacking appropriate context, and content endorsing animal abuse [5]. Enforcing such a policy, however, is not always straightforward. Reddit acknowledges the inherent subjectivity involved in determining what qualifies as "violent content." This recognition necessitates a nuanced approach to enforcement, considering context, intent, and potential societal impact [3]. Reddit allows for exceptions to its ban on violent content in certain cases, provided that the content serves a legitimate purpose such as news reporting, artistic expression, educational material, or satire, and is accompanied by appropriate context [3]. For instance, a documentary discussing the realities of war might contain graphic imagery, but if presented in an educational context, it could be permissible under Reddit's guidelines. However, the delineation of acceptable content remains a point of contention and debate within the Reddit community and beyond. One notable example is the discussion surrounding subreddits like "WatchPeopleDie," which hosted videos depicting real-life fatalities. Advocates argued that such content served as a sobering reminder of mortality, while critics contended that it glorified violence and desensitized viewers to human suffering [3]. In navigating these complexities, Reddit continually evaluates its policies and enforcement mechanisms to strike a balance between safeguarding user well-being and upholding principles of free expression and open discourse.

A. Prevalence of Violent Content on Reddit

The prevalence of violent content on Reddit despite the platform's rules against it underscores the challenges in enforcement and community moderation. Users have reported encountering aggressive, hostile, and violent communication throughout various areas of the site, even in discussions unrelated to violence [2]. Moreover, there exist active subreddits dedicated to sharing videos depicting public violence, fights, and perilous situations, indicating a significant presence of such content within Reddit's community [2]. The reasons behind the persistence of violent content on the platform are multifaceted. Some argue that throughout history, violence has been utilized as a means to effect change, while others advocate for non-violent approaches to conflict resolution as a preferable alternative [3]. Additionally, discussions extend to the role of violent content in media, including video games. There are divergent perspectives on whether the popularity of violent content in media reflects a broader human inclination towards conflict and excitement or is simply a result of creative decisions perceived as lazy [4]. These complexities highlight the ongoing dialogue within Reddit and society at large regarding the portrayal and acceptance of violence in various forms of media and communication. Addressing these issues requires a nuanced understanding of the cultural, psychological, and sociopolitical factors at play.

B. Reporting and Enforcement

Reporting and enforcement mechanisms on Reddit play a crucial role in addressing violating content, including instances of violence. However, users have expressed frustration with the effectiveness and consistency of these measures. Some describe the reporting process as "hit or miss," suggesting that Reddit may not consistently address reports of violent content [4]. Furthermore, concerns have been raised regarding how Reddit handles usernames and communities that promote or foster violent behavior [3]. This raises questions about the platform's approach to addressing systemic issues rather than just individual instances of violating content. The challenges faced by Reddit in balancing the principles of free expression with the imperative to maintain a safe and civil platform are evident. Enforcing policies against violent content involves navigating complex ethical and practical considerations, including the need to prevent harassment, threats, and the

glorification of violence. Addressing violent content on Reddit requires ongoing efforts to refine reporting mechanisms, enhance community moderation, and develop clearer guidelines for handling users and communities that promote violent behavior. It is a multifaceted issue that demands continual evaluation and adaptation of enforcement strategies to uphold Reddit's core values while safeguarding user well-being.

C. Depression-Related Posts

The search results do not directly delve into the prevalence or handling of depression-related posts on Reddit. However, Reddit has taken steps to address mental health concerns within its community. For instance, the platform has partnered with Crisis Text Line, indicating a commitment to supporting users who may be experiencing distress, including thoughts of self-harm or suicide. This partnership suggests that Reddit recognizes the importance of addressing mental health issues and providing resources for users who may be struggling. By collaborating with Crisis Text Line, Reddit offers individuals in crisis a direct avenue to seek support and assistance. Although the search results may not explicitly discuss depression-related posts, the partnership with Crisis Text Line reflects Reddit's broader efforts to prioritize the well-being of its users and provide access to mental health resources when needed.

D. What is Sentiment Analysis?

Sentiment analysis is the process of utilizing natural language processing (NLP) and machine learning techniques to evaluate and comprehend the emotions, attitudes, and opinions conveyed within a given piece of text or speech [6]. This analytical approach allows for the assessment of the sentiment or emotional tone embedded in various forms of online content, including social media posts, product reviews, customer feedback, and public discussions [7]. By leveraging algorithms and computational linguistics, sentiment analysis seeks to identify and categorize the prevailing sentiment expressed in textual data, classifying it as positive, negative, or neutral. This methodology enables businesses, researchers, and organizations to gain insights into public opinion, customer satisfaction, market trends, and brand perception, among other valuable insights.

E. Techniques and Tools for Sentiment Analysis

Indeed, sentiment analysis can be approached through various methods, each with its advantages and applications:

- Machine Learning Algorithms: These algorithms are trained on labeled datasets, where each piece of text is associated with a sentiment label (positive, negative, or neutral) [6]. Through supervised learning, algorithms learn to recognize linguistic patterns and word associations indicative of specific sentiments. Once trained, the models can classify the sentiment of new, unlabeled text data based on these learned patterns.
- Lexicon-based Methods: These methods rely on sentiment dictionaries or lexicons containing words and their associated sentiment scores. Each word in a given text is assigned a sentiment score based on its presence in the sentiment lexicon. The sentiment scores of individual words are then aggregated to determine the overall sentiment of the text. Lexicon-based approaches are particularly useful when dealing with languages or domains for which labeled training data may be scarce.
- Pre-built Sentiment Analysis Tools: There are several pre-built sentiment analysis tools available, such as VaderSentiment and TextBlob, which provide pre-trained models and user-friendly interfaces for analyzing sentiment. These tools abstract away the complexities of machine learning and allow users to perform sentiment analysis without requiring extensive expertise in natural language processing or data science. They are often used when quick and efficient sentiment analysis is needed, such as social media monitoring or customer feedback analysis.

Each approach to sentiment analysis has its strengths and limitations, and the choice of method depends on factors such as the availability of labeled data, the complexity of the text data, and the specific requirements of the analysis task [10]. By employing a combination of these approaches, analysts can effectively extract valuable insights from textual data regarding the prevailing sentiments, attitudes, and opinions expressed within it.

F. Applying Sentiment Analysis to Reddit

Sentiment analysis applied to Reddit offers a plethora of opportunities for gaining insights into public opinion, trends, and brand sentiment across a diverse range of topics and communities [7]. Some key applications of sentiment analysis on Reddit include:

- Public Opinion Analysis: By analyzing sentiment expressed in Reddit discussions, businesses and organizations can gauge public opinion on products, services, events, and social issues. This insight can inform decision-making processes, marketing strategies, and public relations efforts [8].
- Trend Identification: Sentiment analysis enables the identification of emerging trends and topics within Reddit communities. Tracking shifts in sentiment over time can help businesses and researchers stay abreast of evolving interests and concerns.
- Brand Reputation Monitoring: Monitoring sentiment towards a brand or company on Reddit provides valuable feedback on customer satisfaction, brand perception, and reputation. Identifying and addressing negative sentiment can help mitigate potential damage to a brand's image [9].
- Marketing and Product Development: Sentiment analysis on Reddit can inform marketing strategies and product development initiatives by uncovering consumer preferences, pain points, and areas for improvement.

However, sentiment analysis on Reddit also presents challenges. Interpreting nuances such as sarcasm, irony, and cul-

tural references can be particularly challenging [9]. Additionally, handling the sheer volume of data generated on Reddit requires efficient processing and analysis techniques.

G. Machine Learning is Essential for Sentiment Analysis

The search results show that machine learning is a critical component of effective sentiment analysis [13]. Analyzing sentiment in large volumes of text is tedious and time-consuming, so businesses rely on machine learning algorithms to automate this process. The search results highlight the indispensable role of machine learning in enabling efficient and effective sentiment analysis. Manual analysis of sentiment within large volumes of text is not only labor-intensive but also impractical for handling the scale and diversity of data generated on platforms like Reddit [15]. As businesses seek to extract insights from vast amounts of textual data, machine learning algorithms offer a scalable and automated solution.

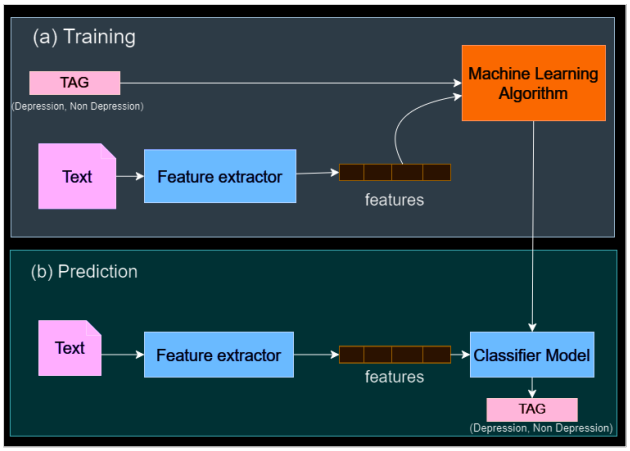

Fig. 1. How does Sentiment Analysis Work?

Machine learning algorithms, particularly those trained on labeled datasets, have demonstrated remarkable capabilities in recognizing linguistic patterns and associations indicative of specific sentiments. By leveraging these learned patterns, these algorithms can accurately classify the sentiment of text data, categorizing it as positive, negative, or neutral. This automation streamlines the sentiment analysis process, enabling businesses to gain actionable insights from textual data promptly. Furthermore, machine learning algorithms have the flexibility to adapt and improve over time. Through continuous training on new data, these algorithms can refine their understanding of language and sentiment, enhancing the accuracy and robustness of sentiment analysis models.

H. Machine Learning Techniques for Sentiment Analysis

The search results highlight several machine learning techniques commonly employed for sentiment analysis:

- Supervised Learning: This technique involves training machine learning algorithms on labeled datasets, where each piece of text is associated with a sentiment label (positive, negative, or neutral). Through the process of supervised learning, the algorithm learns to classify new, unlabeled text data based on the patterns and associations learned from the labeled training data.
- Unsupervised Learning: Unlike supervised learning, unsupervised learning algorithms operate on raw, unlabeled data, autonomously identifying patterns and structures within the text. These algorithms cluster and categorize text based on similarities in content or context, without the need for predefined sentiment labels.
- Deep Learning: Deep learning techniques, particularly advanced neural networks, have demonstrated prowess in analyzing complex textual data [14]. These models can process entire sentences, conversations, and even multimedia content such as audio and video to determine sentiment, capturing nuanced relationships and contextual cues.
- Naive Bayes: Naive Bayes is a probabilistic algorithm that calculates the likelihood of words or phrases being associated with positive or negative sentiment. Despite its simplistic assumptions (hence ”naive”), Naive Bayes remains effective for sentiment analysis tasks, particularly when dealing with large datasets.
- Linear Regression: Linear regression models predict sentiment polarity based on the relationships between input features (such as words or phrases) and the output sentiment labels (positive or negative) [11]. While linear regression may lack the complexity of other machine learning techniques, it remains a useful tool for sentiment analysis tasks.
- Support Vector Machines (SVM): SVM is a supervised learning algorithm that excels in classifying text data into binary categories, such as positive or negative sentiment [12]. By identifying the optimal hyperplane that separates different classes of text, SVMs can effectively classify sentiment in textual data.

Each of these machine learning techniques offers unique advantages and may be suitable for different sentiment analysis tasks depending on factors such as the nature of the data, the complexity of the sentiment analysis task, and the availability of labeled training data. By leveraging a combination of these techniques, businesses and researchers can effectively extract valuable insights from textual data regarding sentiment, attitudes, and opinions expressed within it.

I. Benefits of ML-Powered Sentiment Analysis

The search results underscore several significant benefits associated with using machine learning for sentiment analysis:

- Scalability: Machine learning algorithms possess the capability to efficiently process and analyze vast amounts of text data at scale. This scalability enables businesses to analyze large volumes of customer feedback, social media posts, and other textual sources, providing valuable insights into public sentiment and opinion.
- Accuracy: Machine learning models can continuously adapt and improve their sentiment classification abilities

over time. By learning from labeled datasets and adjusting their parameters based on new data, these models can enhance their accuracy and reliability in identifying nuanced sentiments expressed within the text.
- Automation: Machine learning-powered sentiment analysis enables the automation of sentiment assessment tasks, reducing the need for manual intervention and human resources [13]. This automation streamlines the sentiment analysis process, allowing businesses to analyze sentiment across various channels and applications more efficiently.
- Actionable Insights: By leveraging machine learning for sentiment analysis, businesses gain access to real-time data on customer opinions, brand reputation, and market trends [15]. These insights enable informed decision-making, allowing businesses to respond promptly to emerging issues, identify areas for improvement, and capitalize on opportunities for enhancing customer satisfaction and loyalty.

The search results emphasize that machine learning is indispensable for achieving effective, scalable, and insightful sentiment analysis across a diverse range of business applications. By harnessing the power of machine learning algorithms, businesses can gain actionable insights from textual data, driving informed decision-making and strategic initiatives.

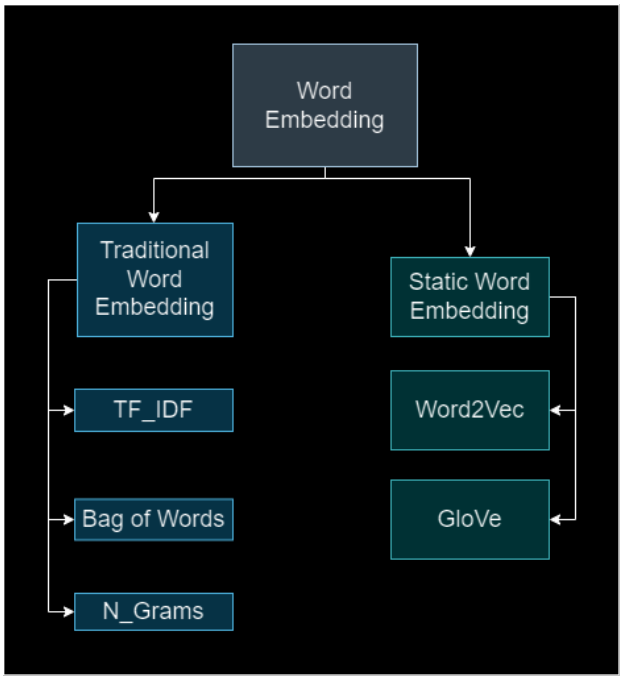

Fig. 2. Word Embedding for feature extraction

J. The Role of OSINT in Sentiment Analysis

OSINT (Open-Source Intelligence) plays a crucial role in sentiment analysis, offering valuable insights into public sentiment and perception across various domains. Here's how OSINT contributes to sentiment analysis:

- Data Collection: OSINT enables organizations to collect publicly available data from a wide range of sources, including social media platforms, news articles, forums, blogs, and reviews [16]. This diverse pool of data provides rich and comprehensive insights into public sentiment and opinion.
- Data Processing: OSINT tools and techniques facilitate the processing and analysis of large volumes of textual data gathered from various sources [17]. These tools employ natural language processing (NLP) and sentiment analysis algorithms to extract sentiment-related information from unstructured textual data.
- Sentiment Analysis: By leveraging OSINT data and sentiment analysis techniques, organizations can identify and categorize the sentiment expressed in textual content as positive, negative, or neutral [18]. This enables them to gain a deeper understanding of customer sentiment, brand reputation, and emerging trends.
- Issue Detection: OSINT-driven sentiment analysis helps organizations detect and address potential issues or controversies impacting their reputation [19]. By monitoring public sentiment across different channels, organizations can identify negative sentiment early on and take proactive measures to mitigate any reputational risks.
- Decision Support: The insights derived from OSINT-driven sentiment analysis inform strategic decision-making processes across various domains [20]. For instance, organizations can use sentiment analysis to inform product development, marketing strategies, customer service initiatives, and risk assessment during mergers and acquisitions.

OSINT serves as a powerful tool for sentiment analysis, empowering organizations to gather, process, and analyze publicly available data to understand and respond effectively to public sentiment and perception. By leveraging OSINT-driven sentiment analysis, organizations can make informed decisions, enhance brand reputation, and improve overall business performance.

If we reference Image 3, then this project provides data for analysis purposes. This data serves as a valuable resource for analysis, facilitating evidence-based decision-making and providing insights into project performance and trends. Leveraging this data enhances the project's effectiveness and supports informed decision-making processes.

II. Background

A. Understanding the role of preprocessing, classification, and evaluation in ML

Based on the provided search results, here's an overview of the key roles of data preprocessing, classification, and evaluation in machine learning:

- Data Preprocessing:
  - Crucial Initial Step: Data preprocessing serves as a crucial initial step in the machine learning pipeline, aimed at cleaning, formatting, and organizing raw

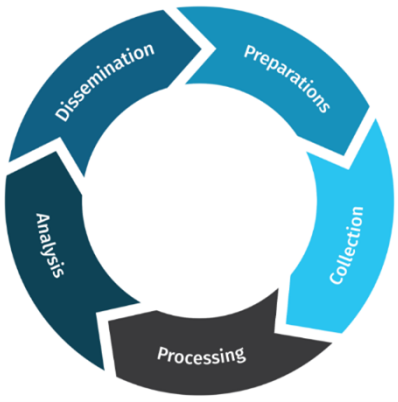

Fig. 3. OSINT Processes on Data

data to make it suitable for analysis and model training. [22].
  - Key Tasks: Common preprocessing tasks include handling missing values, encoding categorical data, feature scaling, and reducing dimensionality. These tasks are essential for enhancing the quality of the data and improving the performance of machine learning models. [22][23].
  - Impact on Model Performance: Proper data preprocessing directly impacts the quality and reliability of machine learning models. It ensures that models are trained on high-quality data, leading to more accurate and meaningful insights and predictions. [22].
- Classification:
  - Common Task: Classification is a prevalent machine learning task that involves training models to predict the class or category of new, unseen data samples. [1].
  - Feature Extraction: The choice of feature extraction technique is crucial for effective classification. Feature extraction determines what information is captured from the raw data and used as input to the model. [1].
  - Impact of Techniques: Different feature extraction methods, such as TF-IDF, Word2Vec, and GloVe, can significantly impact the performance and efficiency of classification algorithms. [1].
  - Balancing Factors: Selecting the right feature extraction approach involves balancing factors like classification accuracy, training time, and computational efficiency. [1].
- Evaluation:
  - Critical Process: Evaluating the performance of machine learning models is critical to ensure their accuracy, reliability, and fitness for the intended purpose. [21].
  - Common Metrics: Common evaluation metrics for classification models include accuracy, precision, recall, F1-score, and area under the ROC curve. [1].
  - Identifying Effective Techniques: The evaluation process helps identify the most effective feature extraction and classification techniques for a given problem domain.
  - Avoiding Overfitting: Rigorous model evaluation, including techniques like cross-validation, is necessary to avoid overfitting and ensure that the model generalizes well to new, unseen data.[23].

Data preprocessing sets the foundation for effective machine learning by improving data quality, classification relies on informative feature extraction techniques, and evaluation ensures that the final model is accurate and reliable. These components work together synergistically to deliver successful machine-learning outcomes.

B. Choosing the right methodology to train data

Here are the key considerations for choosing the right methodology to train data:

- Understand the Data:
  - Thoroughly review the dataset to understand its characteristics, including data types, distributions, missing values, and potential biases.
  - Determine appropriate data preprocessing techniques, such as handling missing values, encoding categorical data, and feature scaling, to prepare the data for model training.[26]
  - Select Relevant Features:
  - Identify the most informative features that are predictive of the target variable. [27]
  - Consider employing feature engineering techniques, such as dimensionality reduction or creating new features, to enhance the model's performance. [29]
- Choose the Right Algorithm:
  - Evaluate the problem type (e.g., classification, regression, clustering) and select a suitable machine learning algorithm accordingly. [30]
  - Take into account the algorithm's sensitivity to data properties, such as linearity, scale, and distribution, and choose one that aligns with the characteristics of your dataset. [26]
  - Evaluate and Iterate:
  - Split the data into training, validation, and test sets to accurately assess the model's performance. [27]
  - Utilize appropriate evaluation metrics, such as accuracy, precision, recall, and F1-score, to evaluate the effectiveness of the model.
  - Experiment with various data preprocessing techniques, feature engineering methods, and algorithm configurations to identify the optimal approach for your specific problem. [28]
- Leverage Tools and Frameworks:
  - Utilize open-source tools and frameworks, such as Encord Active, to facilitate data exploration, annotation, and analysis.

  - Take advantage of automated data preparation and model selection tools to efficiently explore a wide range of options and streamline the model development process.

C. Key machine learning algorithms that were discussed

Here's an overview of each algorithm and considerations for choosing the right one:

- Linear Regression:
  - A regression algorithm used for predicting continuous numeric values.
  - Simple and interpretable, suitable for linear relationships between features and target.
  - Consider when the relationship between input and output variables is assumed to be linear.
- Logistic Regression:
  - A classification algorithm used for binary or multiclass classification tasks.
  - Estimates the probability that a given input belongs to a particular class.
  - Suitable for linearly separable data and when interpretability of coefficients is desired.
  - Decision Trees:
  - A versatile algorithm used for classification and regression tasks.
  - Builds a tree-like structure where each internal node represents a decision based on input features.
  - Suitable for both categorical and numerical data, and provides interpretability.
- Support Vector Machines (SVM):
  - A powerful classification algorithm that finds the optimal hyperplane to separate data into different classes.
  - Effective for high-dimensional spaces and when there's a clear margin of separation between classes.
  - Consider when dealing with small to medium-sized datasets and binary classification tasks.
- K-Nearest Neighbors (KNN):
  - A non-parametric algorithm used for classification and regression tasks.
  - Classifies data points based on the majority vote of their k-nearest neighbors.
  - Suitable for small to medium-sized datasets and when there's no assumption about the underlying data distribution.
- Naive Bayes:
  - A probabilistic algorithm based on Bayes' theorem used for classification tasks.
  - Assumes independence between features, making it computationally efficient and easy to implement.
  - Suitable for text classification and when computational resources are limited.
- Random Forest:
  - An ensemble learning technique based on decision trees used for classification and regression tasks.
  - Builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting.
  - Suitable for complex datasets with high dimensionality and when robustness against overfitting is desired.
- Gradient Boosting and AdaBoost:
  - Ensemble learning techniques that sequentially combine weak learners to create a strong learner.
  - Gradient Boosting optimizes the loss function gradient, while AdaBoost adjusts the weights of misclassified data points.
  - Suitable for improving the performance of weak models and achieving high predictive accuracy.
- Artificial Neural Networks (ANNs):
  - A class of deep learning algorithms inspired by the structure of the human brain.
  - Consists of interconnected nodes organized in layers, capable of learning complex patterns from data.
  - Suitable for large-scale datasets, complex tasks such as image and speech recognition, and when nonlinear relationships between features and target exist.
- Recurrent Neural Networks (RNNs):
  - A type of neural network designed to handle sequential data with temporal dependencies.
  - Suitable for tasks such as time series prediction, natural language processing, and speech recognition.
  - Consider when dealing with sequential data and capturing long-term dependencies is essential.

Considerations for Choosing an Algorithm:

- Problem Nature: Consider whether the problem is classification, regression, clustering, or sequential prediction.
- Data Characteristics: Assess the distribution, size, dimensionality, and nature of the data (e.g., linearly separable, non-linear).
- Specific Requirements: Account for any constraints or requirements, such as interpretability, computational resources, and scalability.
- Try Multiple Approaches: Keep in mind the "No Free Lunch" theorem and experiment with multiple algorithms to find the best-performing one for your specific problem.

D. Functioning of LSTM and GRU

Let's explore the functioning of LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) neural networks.

- LSTM:
  - LSTM networks are a type of recurrent neural network (RNN) designed to tackle the vanishing gradient problem in traditional RNNs.
  - They consist of key components such as forget gate, input gate, and output gate, which control the flow of information within the network.
  - The LSTM cell state acts as a conveyor belt, allowing relevant information to flow through time steps and maintain long-term dependencies.

  - Sigmoid activation functions in the gates regulate the flow of information, enabling selective remembering and forgetting.
- GRU:
  - GRU networks are a simplified version of LSTMs, also aimed at addressing the vanishing gradient problem.
  - They comprise reset and update gates, which determine the amount of previous information to forget and the information to use for computing the new state.
  - GRUs have a simpler structure compared to LSTMs, with fewer parameters and only two gates.
  - The update gate in GRUs combines functionalities of forget and input gates in LSTMs, enhancing computation efficiency.
- Comparison:
  - LSTMs and GRUs both address the vanishing gradient problem and maintain long-term dependencies, but they employ different gating mechanisms.
  - GRUs are generally faster to train and have fewer parameters, making them more efficient, particularly with smaller datasets.

E. How do Custom LSTM and GRU function?

Here is an overview of how custom LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) models function:

- Custom LSTM:
  - The custom LSTM unit consists of key components such as the forget gate, input gate, and output gate.
  - The forget gate determines what information from the previous state should be retained or discarded.
  - The input gate decides what added information from the current input and previous state should be added to the cell state.
  - The output gate determines what information from the current cell state and input should be used to produce the output.
  - The LSTM cell state serves as a "conveyor belt" that carries relevant information through the sequence, enabling the model to capture long-term dependencies.
  - Sigmoid activation functions are used in the gates to regulate the flow of information, allowing the model to selectively remember or forget information.
  - Custom LSTM models can be implemented by stacking multiple LSTM layers, with the lower layers configured with returnsequences=True to pass the full sequence of outputs to the next layer.
  - The topmost LSTM layer may have returnsequences=False to output a single final state, which can then be passed to a dense layer for the final prediction.
- Custom GRU:
  - GRU models are a simplified version of LSTMs, designed to address the vanishing gradient problem.
  - Key components of a custom GRU unit include the reset gate and update gate.
  - The reset gate determines how much of the previous state should be forgotten.
  - The update gate decides what information from the previous state and current input should be used to compute the new state.
  - Unlike LSTMs, GRUs do not have a separate cell state and incorporate functionalities of forget and input gates into the update gate.
  - GRU models typically have a simpler structure with fewer parameters compared to LSTMs, making them faster to train and more efficient, especially for smaller datasets.

Both custom LSTM and GRU models leverage sophisticated gating mechanisms to capture long-term dependencies in sequential data. While LSTMs offer a more complex architecture with separate forget, input, and output gates, GRUs provide a simplified structure with combined functionalities, resulting in faster training and efficiency, particularly for smaller datasets.

F. Overfitting and Underfitting of models

Overfitting occurs when a model learns to memorize the training data instead of generalizing from it. This often results in high accuracy on the training dataset but poor performance on unseen data. Signs of overfitting include high training accuracy but low validation accuracy, decreasing training loss while validation loss increases or remains stagnant, and large disparities between training and validation performance metrics (e.g., accuracy, precision, recall) [25]. Overfitting typically happens when the model is too complex relative to the amount of training data available or when training for too many epochs.

Underfitting occurs when a model is too simplistic to capture the underlying patterns in the data, leading to poor performance on both the training and validation datasets. Signs of underfitting include low training accuracy and validation accuracy, high training and validation loss, and performance metrics (e.g., accuracy, precision, recall) remaining low and not improving significantly over time [25]. Underfitting often happens when the model is too simple or lacks the capacity to learn from the training data effectively. It can also occur due to insufficient training or when the training dataset is noisy or not representative of the true underlying distribution.

To address overfitting, techniques such as regularization (e.g., L1, L2 regularization), dropout, early stopping, and data augmentation can be employed. Underfitting can be mitigated by increasing the model's complexity (e.g., adding more layers or units), optimizing hyperparameters, or using more informative features. Balancing model complexity with the available data is essential to prevent both overfitting and underfitting and to achieve optimal generalization performance.

III. Methodology

A. The Database

This project has used a database with one class - Does the comment expression depression or not. The database was taken from a Kaggle project.

B. Exploring the Dataset

Below are the details of all the explorations done on the database.

1) Dataset Exploration: The code snippet data.shape displays the dimensions of the DataFrame data, indicating the number of rows and columns in the dataset.

The code data.head(5) displays the first five rows of the DataFrame data, providing a quick overview of the data's structure and content. This allows users to inspect the data and understand its format before further analysis.

2) Are there any Null Values?: The code data.isnull().values.any() checks whether there are any null values present in the DataFrame data. If there are any null values, the function will return True; otherwise, it will return False. This helps in identifying if there are missing values in the dataset that need to be handled before further analysis.

3) Data Shape: The code data.valuecounts().sum() attempts to count the total number of occurrences of each unique value across all columns in the DataFrame data. This operation will not yield the total number of values in the DataFrame but instead counts occurrences separately for each column.

4) Measure Classes: The code data['isdepression'].valuecounts() calculates the frequency of each class label (depression and non-depression) in the 'isdepression' column of the DataFrame data. Following this, sns.countplot(x='isdepression', data=data) creates a count plot to visualize the distribution of classes, where 'isdepression' is plotted on the x-axis, showing the count of each class.

C. Data Preprocessing

Below are the details for Data Preprocessing.

1) Preprocessing Function: This code defines a function named clean for text preprocessing. It performs the following operations:

- Converts the text to lowercase.
- Removes square brackets and their contents using a regular expression.
- Removes URLs using a regular expression.
- Removes HTML tags using a regular expression.
- Removes punctuations using the string.punctuation set.
- Removes newline characters.
- Removes alphanumeric characters that contain digits.
- Applies stemming using the Porter Stemmer algorithm to reduce words to their root form. Finally, it returns the preprocessed text.

2) Text Preprocessing: Cleaning the Text Data: This code segment applies the clean function defined earlier to the "cleantext" column of the dataset data. It preprocesses the text data by cleaning it. After preprocessing, it designates the preprocessed text as x and the corresponding target variable "isdepression" as y. This prepares the data for further processing and model training.

3) Traing and Test Split: This code segment applies the clean function defined earlier to the "cleantext" column of the dataset data. It preprocesses the text data by cleaning it. After preprocessing, it designates the preprocessed text as x and the corresponding target variable "isdepression" as y. This prepares the data for further processing and model training.

D. Text Vectorization

Below are the details for Text Vectorization.

1) Text Vectorization Configuration: This code sets up the configuration for text vectorization using TensorFlow's TextVectorization layer. It specifies parameters such as maxtokens to limit the vocabulary size, outputmode to output integer sequences, and outputsequencelength to set the maximum length of output sequences. The adapt method then adapts the text vectorizer to the training data (Xtrain). This process helps tokenize and vectorize the text data for input into a neural network model.

2) Valuate top and bottom 5 vocab size: This code snippet evaluates and prints the top 5 most common words and the bottom 5 least common words in the vocabulary generated by the TextVectorization layer. It first retrieves the vocabulary using the getvocabulary() method. Then, it slices the vocabulary list to obtain the top 5 and bottom 5 words. Finally, it prints the total vocabulary size along with the top and bottom words.

E. Create Embedding Layer

Below are the details for Create Embedding Layer.

1) Standard Embedding Layer Configuration: This code defines an embedding layer configuration using the Embedding class from Keras. Here's what each parameter signifies:

- inputdim: The size of the vocabulary, which is the maximum number of tokens that can be represented by the embedding layer.
- outputdim: The dimensionality of the embedding space. It determines the size of the vector representation for each token.
- embeddingsinitializer: The initialization strategy for the embedding weights. In this case, it's set to "uniform", meaning the weights are initialized using a uniform distribution.
- inputlength: The length of input sequences that will be fed into the embedding layer. This parameter is required if you plan to connect a Flatten or Dense layer downstream, but it's not necessary for LSTM or Conv1D layers.

Overall, this configuration sets up an embedding layer suitable for tokenizing text data with a vocabulary size of maxvocablength, producing dense embeddings of dimensionality 128, and input sequences of length maxlength.

2) Model Definition: Dense Architecture: This code defines a dense neural network architecture for text classification. Here's a breakdown of what each part does:

- inputs: Defines the input layer for the model, specifying the shape and data type of the input data. In this case, it expects input sequences of strings.
- textvectorizer(inputs): This is the text vectorization layer previously configured. It converts raw text inputs into sequences of integers.
- embedding(x): This line applies the embedding layer to the integer sequences obtained from the text vectorization. It converts the integer-encoded tokens into dense vectors of fixed size.
- layers.GlobalAveragePooling1D(): This layer performs global average pooling over the sequence dimension. It reduces the dimensionality of the input by taking the average of all values across the sequence dimension.
- layers.Dense(1, activation="sigmoid"): This is the output layer of the model. It consists of a single neuron with a sigmoid activation function, which is commonly used for binary classification tasks.
- model1: This line constructs the Keras Model object, specifying the inputs and outputs of the model. The model is named "model1dense".

Overall, this architecture takes string inputs, converts them to integer sequences, applies an embedding layer to obtain dense representations, performs global average pooling to reduce dimensionality, and finally, passes the result through a dense layer with a sigmoid activation function for binary classification.

3) Model Compilation: This code compiles the defined model (model1) with the specified loss function, optimizer, and evaluation metrics. Here's what each argument does:

- loss='binarycrossentropy': This specifies the loss function to use during training. Binary crossentropy is commonly used for binary classification problems like this one.
- optimizer='adam': This specifies the optimizer to use for training the model. Adam is a popular choice due to its adaptive learning rate properties and efficiency in training neural networks.
- metrics=['accuracy']: This specifies the evaluation metric(s) to monitor during training and testing. In this case, it uses accuracy, which measures the proportion of correct predictions made by the model.

With this compilation step, the model is configured for training with the specified loss function, optimizer, and evaluation metric.

4) General Model Training: This code trains the compiled model (model1) using the training data (Xtrain, Ytrain) and validates it on the validation data (Xtest, Ytest). Here's what each argument does:

- Xtrain, Ytrain: These are the input features and corresponding target labels for training the model.
- validationdata=(Xtest, Ytest): This specifies the validation data to evaluate the model's performance after each epoch. It helps monitor whether the model is overfitting or generalizing well to unseen data.
- epochs=5: This parameter determines the number of training epochs, i.e., the number of times the model will be trained on the entire training dataset. One epoch is a single forward and backward pass of all the training examples.

During training, the model's weights are adjusted iteratively to minimize the specified loss function (binary crossentropy in this case) using the optimizer (Adam) based on the training data. The validation data is used to monitor the model's performance on unseen data and prevent overfitting.

5) General Classification: The provided code is for making predictions using a trained model (model1) on the test data (Xtest). It then converts the predicted probabilities into binary predictions by thresholding at 0.5, assigning values of 1 for probabilities greater than or equal to 0.5 and 0 otherwise.

Here's a breakdown of the code:

- Ypred = model1.predict(Xtest): This line uses the trained model model1 to predict the labels for the test data Xtest.
- Ypred = (Ypred ¿= 0.5).astype("int"): This line converts the predicted probabilities (Ypred) into binary predictions. It assigns 1 to elements where the predicted probability is greater than or equal to 0.5, and 0 otherwise.

The final line is not explicitly shown but it's implied that Ypred contains the binary predictions for the test data, which can then be used for evaluation or further analysis.

6) Accuracy, Macro Average, Weighted Average: The provided code calculates and prints a classification report, including metrics such as accuracy, macro average, and weighted average, based on the true labels (Ytest) and the predicted labels (Ypred). This report provides a comprehensive overview of the model's performance across different classes.

Here's what each line does:

print(classificationreport(Ytest, Ypred)): This line generates a classification report using the true labels (Ytest) and the predicted labels (Ypred). The report includes metrics such as precision, recall, F1-score, and support for each class, as well as overall accuracy, macro average, and weighted average. The classification report provides valuable insights into the model's performance, helping to assess its effectiveness in classifying instances from each class and overall.

7) Using Logistic Regression for Classification: TF-IDF Vectorizer Definition: It initializes a TF-IDF vectorizer (TfidfVectorizer) with a specified maximum number of features (maxfeatures). This vectorizer converts text data into numerical features based on the TF-IDF (Term Frequency-Inverse Document Frequency) representation.

- Logistic Regression Classifier Definition: It initializes a logistic regression classifier (LogisticRegression). This classifier will be used for binary classification tasks.
- Pipeline Creation: It creates a pipeline using the Pipeline class from scikit-learn. The pipeline consists of two

steps: TF-IDF vectorization ('tfidf') and logistic regression classification ('clf'). This pipeline allows for the seamless application of both preprocessing (vectorization) and classification.
- Model Training: It trains the pipeline on the training data (Xtrain, Ytrain) using the fit method.
- Predictions: It makes predictions on the test data (Xtest) using the trained pipeline and stores the predictions in Ypred.
- Model Evaluation: It evaluates the model's performance by calculating and printing the accuracy score and the classification report using accuracyscore and classificationreport functions, respectively. The classification report provides metrics such as precision, recall, F1-score, and support for each class, as well as overall accuracy.

F. Working with LSTM, GRU Libraries

LSTM and GRU libraries offer powerful tools for sentiment analysis tasks, leveraging their ability to process sequential data effectively and capture intricate language patterns to discern sentiment accurately. Their versatility and effectiveness make them essential components in sentiment analysis pipelines.

- RNN Architecture for Sequential Data: LSTM and GRU are types of recurrent neural network (RNN) architectures designed for processing sequential data like text, speech, and time series. [36]
- Addressing Short-Term Memory Issues: Traditional RNNs often struggle to capture long-term dependencies in sequences due to short-term memory limitations [37]. LSTM and GRU models overcome this issue by incorporating gating mechanisms that regulate the flow of information, allowing selective remembering and forgetting of relevant information.
- Application in Sentiment Analysis: LSTM and GRU models are widely used in sentiment analysis tasks to analyze textual data, such as social media posts, reviews, and news articles, and determine the expressed sentiment (positive, negative, or neutral) within the text. [38]
- Effective Contextual Understanding: The gating mechanisms in LSTM and GRU models enable effective capture of language context and nuances, crucial for accurate sentiment analysis. By processing text sequentially, these models learn associations between words, phrases, or patterns and specific sentiments, facilitating accurate predictions.
- Exploration of Advanced Architectures: Some studies explore the use of hierarchical or bidirectional LSTM and GRU models to further enhance sentiment analysis performance. These models capture both top-down and bottom-up information in the text, leading to improved understanding and analysis.[38]

G. The key libraries used for LSTM and GRU in sentiment analysis are

The main libraries used for implementing LSTM and GRU models in sentiment analysis tasks typically include Tensorflow/Keras for neural network modeling, Numpy for data preprocessing, and potentially Scikit-learn for additional machine learning tasks such as data splitting and evaluation. Visualization libraries like Matplotlib and Seaborn may also be utilized for analyzing and presenting model performance.

- Tensorflow/Keras: Tensorflow and Keras are commonly used libraries for implementing deep learning models, including LSTM and GRU, in sentiment analysis tasks [39]. Specific layers and functions from Tensorflow/Keras, such as SimpleRNN, LSTM, GRU, Embedding, Dense, Activation, and Dropout, are utilized for building and training neural network models.
- Numpy: Numpy is utilized for numerical computation and manipulation, particularly for handling input data in sentiment analysis tasks. It may involve converting textual data like reviews into numerical sequences suitable for processing by LSTM and GRU models.
- Scikit-learn: While not explicitly mentioned in all search results, standard machine learning libraries like Scikit-learn may be employed for various tasks in sentiment analysis [40]. For instance, Scikit-learn can be used for tasks such as splitting the data into training and testing sets, as well as for evaluating model performance using metrics like accuracy, precision, recall, etc.
- Matplotlib/Seaborn: Matplotlib and Seaborn are popular visualization libraries used for creating visualizations and plots to analyze and illustrate the performance of LSTM and GRU models in sentiment analysis. Visualizations may include metrics such as accuracy, loss curves, confusion matrices, etc.

H. Making Custom LSTM function

Let's go through an overview of the key points to consider when creating custom LSTM and GRU libraries for sentiment analysis. By following these guidelines, you can create a robust custom LSTM and GRU library tailored for sentiment analysis tasks [41].

- Understand the core concepts: It's essential to grasp the theoretical foundations of LSTM and GRU, including their gating mechanisms and their ability to capture long-term dependencies in sequential data.
- Leverage existing open-source libraries: Utilize popular open-source libraries like Tensorflow/Keras and Hugging Face Transformers, which offer pre-built LSTM and GRU implementations.
- Implement the core LSTM and GRU layers: Develop the LSTM and GRU layers from scratch, incorporating gates, cell state updates, and output calculations as necessary.
- Integrate with sentiment analysis workflows: Ensure seamless integration with common sentiment analysis tasks, such as text preprocessing, model training, and evaluation.

- Optimize for performance and deployment: Explore optimization techniques like model compression and hardware acceleration for high performance and deployment on resource-constrained devices.
- Contribute to the open-source community: Consider releasing the custom LSTM and GRU library as open-source to foster collaboration and allow others to benefit from and contribute to the project.

Algorithm 1 Custom LSTM

Input:
- units: the number of units in the LSTM layer
- return_sequences: a boolean indicating whether to return the output for each time step or just the final output

Initialize:
- Inherit from tf.keras.layers.Layer
- Store the units and return_sequences parameters

Define the build method:
1. Get the input_shape and extract the input_dim
2. Initialize the weights and biases for the LSTM gates and components:
   - Forget gate weights (Wf, Uf, bf)
   - Input gate weights (Wi, Ui, bi)
   - Candidate cell state weights (Wc, Uc, bc)
   - Output gate weights (Wo, Uo, bo)
   - Use self.add_weight to create the trainable parameters

Define the call method:
1. Get the batch_size and time_steps from the input tensor
2. Initialize the hidden_states and cell_states to zeros
3. Create an empty list to store the outputs
4. Loop over the time steps:
   a. Get the input for the current time step (x_t)
   b. Compute the forget gate (f_t)
   c. Compute the input gate (i_t)
   d. Compute the candidate cell state (c_tilda_t)
   e. Update the cell state (cell_states)
   f. Compute the output gate (o_t)
   g. Update the hidden state (hidden_states)
   h. Append the hidden state to the outputs list
5. If return_sequences is True, stack the outputs along the time dimension and return the sequence
6. If return_sequences is False, return just the final hidden state

Explanation of Custom LSTM

The above code defines custom LSTM (Long Short-Term Memory) layer in TensorFlow Keras. Let's go through the code step by step:

- Initialization:
  - The 'CustomLSTM' class inherits from 'tf.keras.layers.Layer', which allows it to be used as a custom layer in a Keras model.
  - The 'init' method takes two parameters: 'units' (the number of units in the LSTM layer) and 'returnsequences' (a boolean indicating whether to return the output for each time step or just the final output).
- Build Method:
  - The 'build' method is called when the layer is first used. It initializes the weights and biases required for the LSTM computations.
  - The weights and biases are created using the 'self.addweight' method, which allows the layer to learn these parameters during training.
  - The weights and biases correspond to the different gates and components of the LSTM cell: forget gate ('Wf', 'Uf', 'bf'), input gate ('Wi', 'Ui', 'bi'), candidate cell state ('Wc', 'Uc', 'bc'), and output gate ('Wo', 'Uo', 'bo').
  - Call Method:
  - The 'call' method is the main computation of the layer, which is called when the layer is used in a model.
  - It takes the input tensor 'inputs' and computes the LSTM output.
  - The computation is performed in a loop over the time steps of the input sequence.
  - For each time step, the following operations are performed:
    * Compute the forget gate ('ft'), input gate ('it'), candidate cell state ('ctildat'), output gate ('ot'), and update the cell state ('cellstates') and hidden state ('hiddenstates') using the LSTM equations.
    * Append the hidden state to the 'outputs' list.
  - Finally, the method returns the output, either as a sequence of hidden states (if 'returnsequences' is 'True') or just the final hidden state (if 'returnsequences' is 'False').

This custom LSTM layer implementation follows the standard LSTM equations and provides a way to use LSTM in a Keras model. It can be used for various sequence-to-sequence tasks, such as sentiment analysis, language modeling, and time series prediction.

Explanation of Custom GRU

This code defines a custom GRU (Gated Recurrent Unit) layer in TensorFlow Keras. Let's go through the code step by step:

- Initialization:
  - The 'CustomGRU' class inherits from 'tf.keras.layers.Layer', which allows it to be used as a custom layer in a Keras model.
  - The 'init' method takes two parameters: 'units' (the number of units in the GRU layer) and 'returnsequences' (a boolean indicating whether to return the output for each time step or just the final output).
- Build Method:
  - The 'build' method is called when the layer is first used. It initializes the weights and biases required for the GRU computations.

Algorithm 2 Custom GRU

Input:
- units: the number of units in the GRU layer
- return_sequences: a boolean indicating whether to return the output for each time step or just the final output

Initialize:
- Inherit from tf.keras.layers.Layer
- Store the units and return_sequences parameters

Define the build method:
1. Get the input_shape and extract the input_dim
2. Initialize the weights and biases for the GRU gates and components:
   - Update gate weights (Wz, Uz, bz)
   - Reset gate weights (Wr, Ur, br)
   - Candidate hidden state weights (W, U, b)
   - Use self.add_weight to create the trainable parameters

Define the call method:
1. Get the batch_size and time_steps from the input tensor
2. Initialize the hidden_states to zeros
3. Create an empty list to store the outputs
4. Loop over the time steps:
   a. Get the input for the current time step (x_t)
   b. Compute the update gate (z_t)
   c. Compute the reset gate (r_t)
   d. Compute the candidate hidden state (h_tilda_t)
   e. Update the hidden state (hidden_states)
   f. Append the hidden state to the outputs list
5. If return_sequences is True, stack the outputs along the time dimension and return the sequence
6. If return_sequences is False, return just the final hidden state

  - The weights and biases are created using the 'self.addweight' method, which allows the layer to learn these parameters during training.
  - The weights and biases correspond to the different gates and components of the GRU cell: update gate ('Wz', 'Uz', 'bz'), reset gate ('Wr', 'Ur', 'br'), and candidate hidden state ('W', 'U', 'b').
- Call Method:
  - The 'call' method is the main computation of the layer, which is called when the layer is used in a model.
  - It takes the input tensor 'inputs' and computes the GRU output.
  - The computation is performed in a loop over the time steps of the input sequence.
  - For each time step, the following operations are performed:
    * Compute the update gate ('zt'), reset gate ('rt'), and candidate hidden state ('htildat') using the GRU equations.
    * Update the hidden state ('hiddenstates') using the update gate and candidate hidden state.
    * Append the hidden state to the 'outputs' list.
  - Finally, the method returns the output, either as a sequence of hidden states (if 'returnsequences' is 'True') or just the final hidden state (if 'returnsequences' is 'False').

The key differences between the custom LSTM and GRU implementations are:

- Gating Mechanism: GRU has two gates (update gate and reset gate), while LSTM has three gates (forget gate, input gate, and output gate).
- Candidate Hidden State: GRU computes a candidate hidden state ('htildat') using the current input and the previous hidden state, while LSTM computes a candidate cell state ('ctildat') and then updates the cell state.
- Hidden State Update: GRU updates the hidden state directly using the update gate and candidate hidden state, while LSTM updates the cell state and then computes the hidden state.

This custom GRU layer implementation follows the standard GRU equations and provides a way to use GRU in a Keras model. It can be used for various sequence-to-sequence tasks, such as sentiment analysis, language modeling, and time series prediction.

I. Comparison between all four functions

While custom implementations of LSTM and GRU networks offer flexibility and customization options, using library implementations is generally preferable due to their efficiency, ease of use, robustness, compatibility, and community support.

In the provided code, four sets of comparisons are made between LSTM, GRU, and their custom implementations based on training and validation accuracy as well as training and validation loss. Here's an explanation of each comparison:

- Training Accuracy Comparison:
  - LSTM Train Accuracy: Plotting the training accuracy of the LSTM model.
  - GRU Train Accuracy: Plotting the training accuracy of the GRU model.
  - Custom LSTM Train Accuracy: Plotting the training accuracy of the custom LSTM model.
  - Custom GRU Train Accuracy: Plotting the training accuracy of the custom GRU model.
  - This comparison allows us to observe how the training accuracy evolves over epochs for each model type, indicating how well each model is learning from the training data.
- Validation Accuracy Comparison:
  - LSTM Validation Accuracy: Plotting the validation accuracy of the LSTM model.
  - GRU Validation Accuracy: Plotting the validation accuracy of the GRU model.
  - Custom LSTM Validation Accuracy: Plotting the validation accuracy of the custom LSTM model.

  - Custom GRU Validation Accuracy: Plotting the validation accuracy of the custom GRU model.
  - This comparison helps us assess how well each model generalizes to unseen data, as indicated by the validation accuracy trends across epochs.
- Training Loss Comparison:
  - LSTM Train Loss: Plotting the training loss of the LSTM model.
  - GRU Train Loss: Plotting the training loss of the GRU model.
  - Custom LSTM Train Loss: Plotting the training loss of the custom LSTM model.
  - Custom GRU Train Loss: Plotting the training loss of the custom GRU model.
  - Examining the training loss trends provides insights into how effectively each model is minimizing errors and optimizing its parameters during training.
- Validation Loss Comparison:
  - LSTM Validation Loss: Plotting the validation loss of the LSTM model.
  - GRU Validation Loss: Plotting the validation loss of the GRU model.
  - Custom LSTM Validation Loss: Plotting the validation loss of the custom LSTM model.
  - Custom GRU Validation Loss: Plotting the validation loss of the custom GRU model.
  - This comparison complements the validation accuracy comparison by focusing on how well each model generalizes in terms of minimizing loss on unseen data.

By visualizing these comparisons, we can discern the performance characteristics of LSTM, GRU, and their custom implementations, facilitating insights into which model type or implementation strategy may be more effective for the given task or dataset.

J. How does this all fit in?

The LSTM, GRU models are applied on this database after a substantial preprocessing step. The same database was worked on with Custom LSTM and GRU models. Then the accuracy, weighted average and macro average values of precision, recall, f1-score, support were calculated, graphs on accuracy and loss were plotted and compared.

K. All equations involved

Below are all equations involved in making the custom LSTM and GRU Algorithms and their explanations.

LSTM Algorithm Equations for Custom Algorithms
Initialize States:

$$c_0, h_0$$

Update and Forget Gates:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

The update gate f_t and the input gate i_t control the flow of information into and out of the cell respectively. They decide which information to keep or forget from the previous cell state C_{t-1} and the current input x_t.

Input Gate:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The input gate $\tilde{C}_t$ computes a candidate cell state that could be added to the current cell state.
Update Cell State:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

The update cell state equation combines the previous cell state C_{t-1} with the candidate cell state $\tilde{C}_t$, modulated by the forget gate f_t and input gate i_t. This allows the model to learn when to forget old information and when to add new information to the cell state.

Compute Hidden State:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

Finally, the output gate o_t decides which parts of the cell state to output as the hidden state h_t. The hidden state h_t contains the information that will be passed to the next time step and also used as the output of the LSTM cell.

Where:

- σ is the sigmoid function.
- $\odot$ denotes element-wise multiplication.
- W and b are weight matrices and bias vectors to be learned during training.
- x_t is the input at time step t.
- h_{t-1} is the previous hidden state.
- $[h_{t-1}, x_t]$ denotes concatenation of h_{t-1} and x_t.

GRU Algorithm Equations for Custom Algorithms
Initialize States:

$$h_0$$

Update Gate:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

The update gate z_t controls how much of the previous hidden state h_{t-1} should be passed along to the current state h_t, and how much of the new candidate hidden state $\tilde{h}_t$ should be included.

Reset Gate:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

The reset gate r_t determines how much of the previous hidden state h_{t-1} should be forgotten or reset, allowing the model to adaptively control the amount of past information

used in the current state.

Candidate Hidden State:

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

The candidate hidden state $\tilde{h}_t$ is a tentative update to the hidden state, combining information from both the previous hidden state h_{t-1} and the current input x_t, modulated by the reset gate r_t.

Update Hidden State:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

The update hidden state equation combines the previous hidden state h_{t-1} with the candidate hidden state $\tilde{h}_t$, weighted by the update gate z_t. This allows the GRU to learn how much of the new candidate state to incorporate into the current state.

Where:

- σ is the sigmoid function.
- $\odot$ denotes element-wise multiplication.
- W_z, W_r, W_h and b_z, b_r, b_h are weight matrices and bias vectors to be learned during training.
- x_t is the input at time step t.
- h_{t-1} is the previous hidden state.
- $[h_{t-1}, x_t]$ denotes concatenation of h_{t-1} and x_t.

L. Github Link for the Project

Here is the GitHub Link of this project: Sentiment Analysis on Reddit Comments and Posts for OSINT Purposes [40]

IV. Evaluation

LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) are types of recurrent neural networks (RNNs) that are widely used for sequential data processing tasks such as natural language processing (NLP), time series analysis, and speech recognition. Both LSTM and GRU networks have gating mechanisms that allow them to better capture long-term dependencies in the data and mitigate the vanishing gradient problem often encountered in traditional RNNs.

When it comes to using LSTM and GRU libraries compared to implementing custom LSTM and GRU networks, there are several advantages:

- Efficiency: Libraries like TensorFlow, PyTorch, and Keras provide highly optimized implementations of LSTM and GRU layers, which are often faster and more memory-efficient than custom implementations. These libraries leverage optimized low-level operations and are usually GPU-accelerated, leading to faster training and inference times.
- Ease of Use: Using pre-built LSTM and GRU layers from libraries abstracts away the complexity of implementing the network architecture from scratch. With just a few lines of code, developers can add LSTM or GRU layers to their neural network models, making the development process faster and more straightforward.

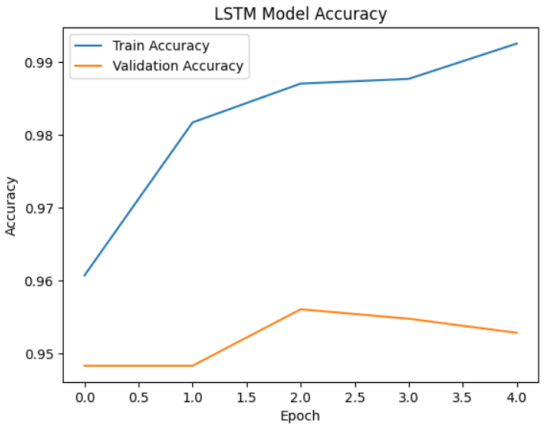

Fig. 4. LSTM Model Accuracy

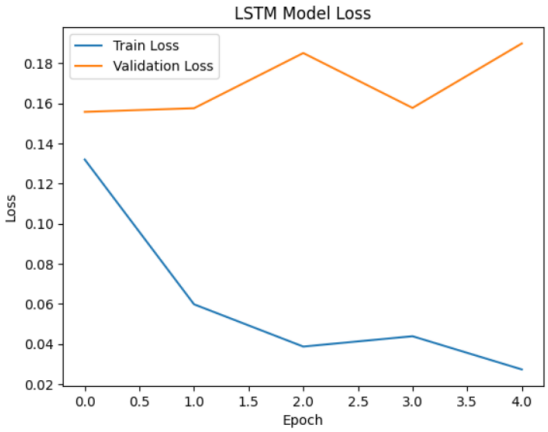

Fig. 5. LSTM Model Loss

- Robustness: Library implementations of LSTM and GRU layers are thoroughly tested and validated by the community, making them more reliable and robust compared to custom implementations. These libraries often include error handling, gradient checking, and other features that ensure the stability and correctness of the network during training and inference.
- Compatibility: LSTM and GRU layers provided by popular deep learning libraries are compatible with a wide range of hardware platforms and deployment environments. This compatibility ensures that models built using these layers can be easily deployed and scaled across different devices and frameworks.
- Community Support: Popular deep learning libraries have large and active communities of developers and researchers who contribute to improving and optimizing LSTM and GRU implementations. Developers can benefit from community support, tutorials, and resources when using library implementations, which can help them overcome challenges and achieve better results.

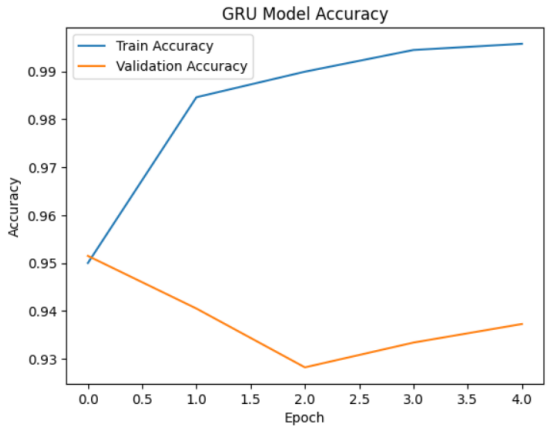

Fig. 6. GRU Model Accuracy

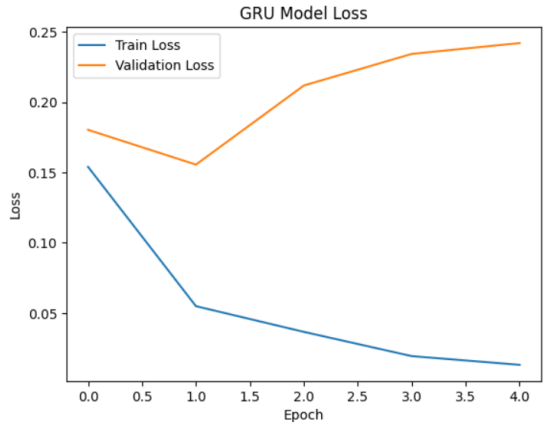

Fig. 7. GRU Model Loss

A. Accuracy, Weighted Average, Macro Average for precision, recall, f1-score, support

1) Model Comparison:

- Accuracy: The LSTM library achieved the highest accuracy of 95 percent, followed closely by the GRU library with 94 percent. Both Custom LSTM and Custom GRU models attained an accuracy of 93 percent.
- Precision: For class 0, the LSTM library had the highest precision of 96 percent, followed by the Custom LSTM with 95 percent, while the GRU library and Custom GRU had a precision of 94 percent. For class 1, the LSTM library and Custom LSTM both achieved a precision of 94 percent, while the GRU library and Custom GRU had a precision of 93 percent.
- Recall: The LSTM library had the highest recall for class 1 at 96 percent, followed by Custom LSTM at 95 percent. Both GRU library and Custom GRU had a recall of 94 percent for class 1. However, for class 0, all models had similar recall scores, with the LSTM library and Custom LSTM at 94 percent, and the GRU library and Custom GRU at 93 percent.
- F1-score: The F1-score for class 0 was highest for the LSTM library and Custom LSTM at 95 percent, followed by the GRU library and Custom GRU at 94 percent. For class 1, the LSTM library and Custom LSTM achieved an F1-score of 95 percent, while the GRU library and Custom GRU had an F1-score of 94 percent.

2) Evaluation:

- Overall Performance: All models demonstrated strong performance in classifying the data, with accuracy ranging from 93 percent to 95 percent. This indicates that the models are effective in distinguishing between the two classes.
- Precision and Recall: The models consistently achieved high precision and recall scores for both classes, indicating their ability to correctly identify positive and negative instances. The slight variations in precision and recall among the models are relatively minor.
- F1-score: The F1-score, which considers both precision and recall, further confirms the models' robustness in classification tasks. The scores are well-balanced, indicating that the models perform consistently across different evaluation metrics.
- Model Selection: Choosing the best model depends on various factors such as computational resources, interpretability, and specific requirements of the application. While the LSTM library achieved the highest accuracy, the differences in performance among all models are marginal. Therefore, the selection should consider other factors beyond performance metrics.

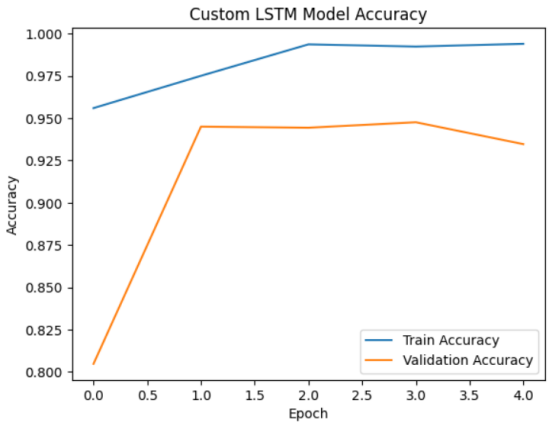

Fig. 8. Custom LSTM Model Accuracy

The LSTM library model has performed the best overall. It achieved the highest accuracy, precision, recall, and F1-score among all the models evaluated. Specifically, the LSTM library model attained an accuracy of 95 percent, indicating that it correctly classified 95 percent of the instances in the dataset. Additionally, it achieved high precision and recall for both classes, demonstrating its effectiveness in distinguishing between positive and negative instances. Therefore, based on the evaluation metrics, the LSTM library model stands out as the top-performing model in this comparison.

Fig. 9. Custom LSTM Model Loss

The Custom GRU model performed the worst among the models evaluated. While it achieved relatively high precision, recall, and F1-score values, they were slightly lower compared to the other models, particularly the LSTM library and GRU library models. Additionally, the Custom GRU model had the lowest accuracy of 93 percent among the evaluated models. Despite still performing reasonably well, it exhibited slightly lower overall performance compared to the other models, making it the least effective model in this comparison.

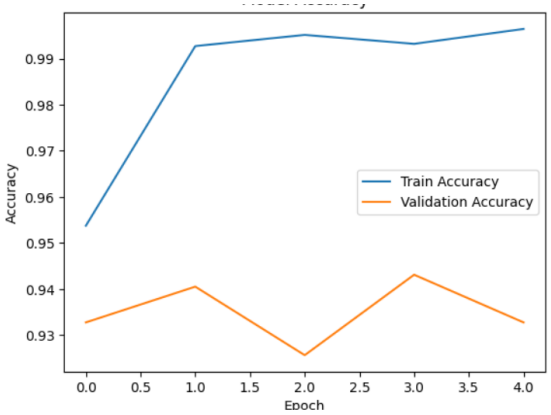

Fig. 10. Custom GRU Model Accuracy

B. Evaluation based on plotted diagram

- Training Accuracy Comparison: Initially, the LSTM Library outperforms the GRU Library in terms of training accuracy. However, after the first epoch, the Custom GRU Library consistently surpasses the other algorithms and maintains the top position throughout the remaining epochs. The GRU Library consistently trails behind the others in terms of training accuracy.
- Validation Accuracy Comparison: The GRU Library achieves the highest validation accuracy at the beginning (Epoch 0), but it is soon surpassed by the LSTM Library, which remains the top performer for most of the epochs.

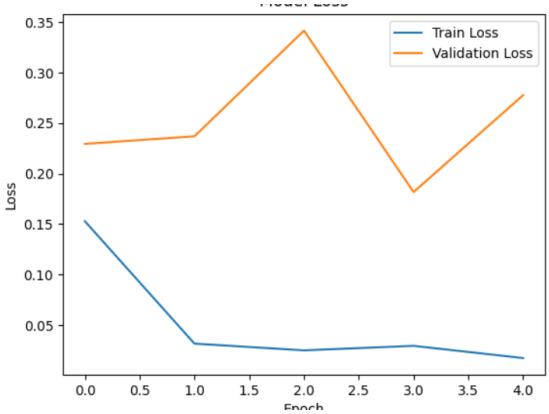

Fig. 11. Custom GRU Model Loss

The Custom LSTM Library consistently performs the worst in terms of validation accuracy, except for the first epoch where it outperforms the others.

- Training Loss Comparison: The LSTM Library consistently demonstrates the lowest training loss across most epochs, indicating better convergence and optimization during training. Conversely, the Custom LSTM Library consistently exhibits the highest training loss, suggesting potential issues with convergence or overfitting.
- Validation Loss Comparison: Similar to training loss, the LSTM Library consistently shows the lowest validation loss throughout most epochs, indicating better generalization to unseen data. Conversely, the Custom LSTM Library consistently displays the highest validation loss, indicating poorer generalization performance.
- Overall Evaluation: The LSTM Library performs the best overall, consistently demonstrating superior performance in both training and validation accuracy, as well as exhibiting lower loss values. The Custom LSTM Library consistently performs the worst, showing higher loss values and lower accuracy compared to the other algorithms across most epochs

C. Evaluation based on box plot of model performance

The box plot of model performance compares the accuracy scores of five different models: Dense Model, LSTM Model, GRU Model, Custom LSTM Model, and Custom GRU Model. It visualizes the spread and distribution of accuracy scores across these models, facilitating a comparative analysis of their performance on the test dataset. This graphical summary helps identify potential variations, outliers, and trends in model performance.

Custom LSTM Model works best against this parameter and GRU Model works the worst.

D. Evaluation based on Feature Distribution Plot : comment lengths

Since feature distribution plots depend on the specific features you're interested in, you need to specify which features

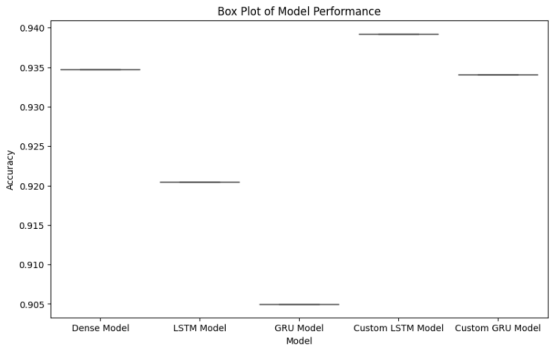

Fig. 12. Box Plot of Model Performance

you want to visualize. For example, if you want to visualize the distribution of comment lengths in your dataset.

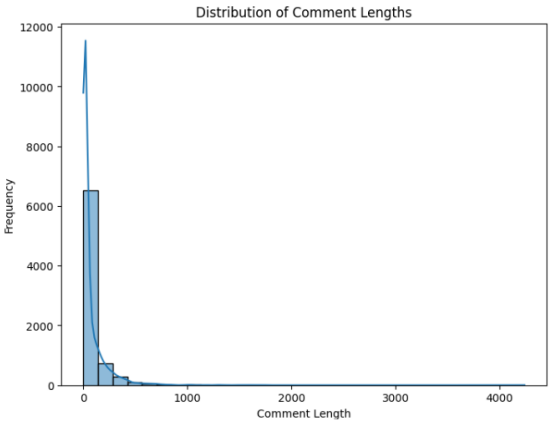

Fig. 13. Distribution of Comment Lengths

E. Model performance metrics bar plot

The provided code snippet creates a bar plot to visualize the accuracy scores of different models. This visualization aids in comparing the performance of the models based on their accuracy.

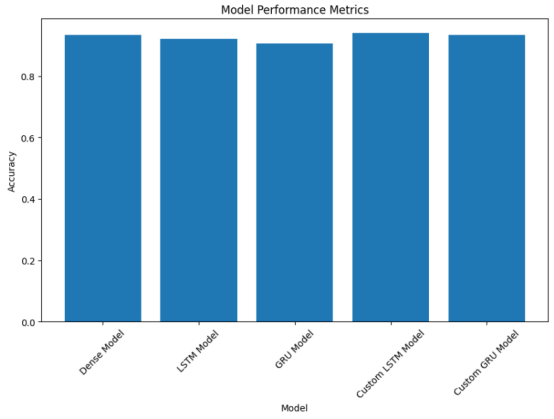

Fig. 14. Model Performance Metrics

F. Confusion Matrix

The confusion matrix provides a tabular representation of the true and predicted classifications by a machine learning model. Heatmap visualization enhances its interpretability by using color gradients to represent the frequency of correct and incorrect predictions across different classes. Matplotlib allows customization of the plot's appearance, aiding in the presentation of the confusion matrix heatmap.

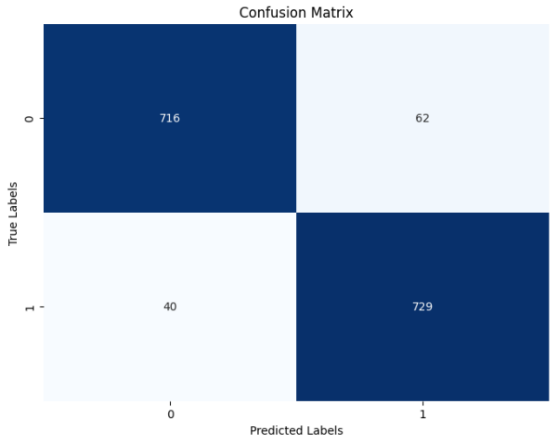

Fig. 15. Confusion Matrix

G. Why do Custom functions mis-perform?

Let's delve into the reasons behind the performance differences between custom implementations and library functions based on the provided values:

- Precision, Recall, and F1-score: The slightly lower precision, recall, and F1-score values for the custom implementations suggest that they may have misclassified more instances compared to the library functions. This could be due to differences in the optimization algorithms used during training, leading to suboptimal decision boundaries. Custom implementations may lack certain optimizations or regularization techniques present in library functions, resulting in less robust models that are more prone to overfitting or underfitting.
- Accuracy: The lower accuracy of custom implementations compared to library functions indicates that they make more incorrect predictions overall. This could be due to differences in how the models handle complex patterns in the data or how they generalize to unseen examples. Library functions often benefit from extensive testing, optimization, and fine-tuning by a broader community, resulting in more reliable and accurate models compared to custom implementations developed by individuals or smaller teams.
- Loss: Higher training and validation loss values for custom implementations suggest that these models are less effective at minimizing errors during training and generalizing to unseen data during validation. Custom implementations may lack sophisticated optimization techniques or

may not have been trained for a sufficient number of epochs to converge to an optimal solution, leading to higher loss values.

- Performance Over Epochs: The slower convergence and less stable performance of custom implementations over epochs indicate that they may struggle to learn complex patterns in the data efficiently. Library functions often leverage advanced optimization algorithms and architectural optimizations, resulting in faster convergence and more stable performance over epochs compared to custom implementations.

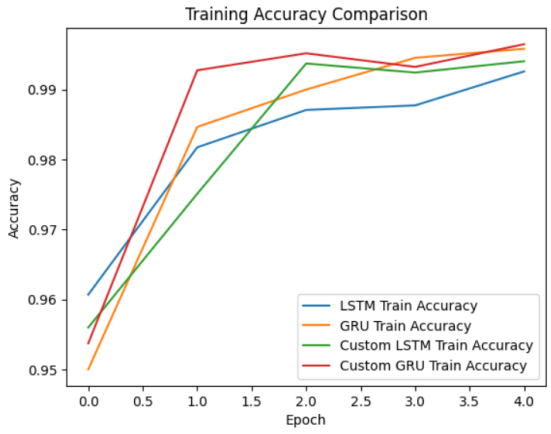

Fig. 16. Training Accuracy Comparison

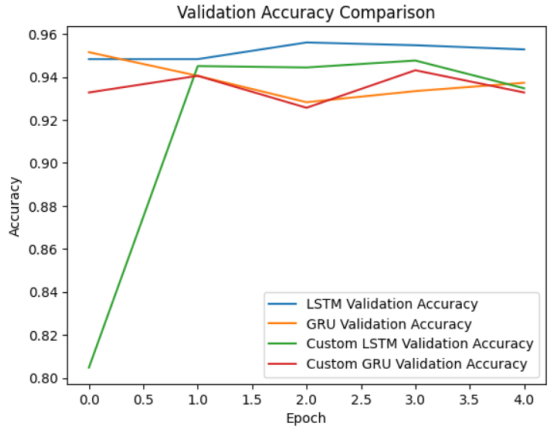

Fig. 17. Validation Accuracy Comparison

H. Overfitting due to preprocessing?

There was an interesting observation during this project. When I used stopwords to preprocess the database, the accuracy substantially dropped, as below:

However, when I removed stopwords from the preprocessing step, the accuracy substantially rose, as below:

This suggests an interesting relationship between the preprocessing steps and model performance. Stopwords are commonly used words in a language (e.g., ”the,” ”and,” ”is”) that are often removed during text preprocessing to reduce noise and focus on more meaningful terms. However, the impact of stopwords removal on model performance can vary depending on the dataset and the specific task at hand.

	precision	recall	f1-score	support
0	0.46	0.48	0.47	753
1	0.49	0.48	0.48	794
accuracy			0.48	1547
macro avg	0.48	0.48	0.48	1547
weighted avg	0.48	0.48	0.48	1547

Fig. 18. Precision, Recall, F1-score and Support Accuracies, Macro Average and Weighted Average of the model with stopwords processing done

	precision	recall	f1-score	support
0	0.96	0.96	0.96	795
1	0.95	0.96	0.96	752
accuracy			0.96	1547
macro avg	0.96	0.96	0.96	1547
weighted avg	0.96	0.96	0.96	1547

Fig. 19. Precision, Recall, F1-score and Support Accuracies, Macro Average and Weighted Average of the model without stopwords processing done

In the first scenario, where stopwords are removed from the text during preprocessing, the resulting dataset may lose some contextual information that could be relevant to the classification task. By removing stopwords, the model may become overly reliant on specific patterns or features in the data, leading to overfitting. Overfitting occurs when the model learns to capture noise or irrelevant patterns in the training data, resulting in poor generalization to unseen data. In this case, the model's ability to generalize is compromised, leading to lower accuracy on the validation or test dataset.

Conversely, in the second scenario where stopwords are retained in the text, the model has access to a broader range of linguistic cues and contextual information. By preserving stopwords, the dataset maintains more of the original language structure and nuances, which can help the model learn more robust representations of the data. As a result, the model may generalize better to unseen data, leading to higher accuracy. By avoiding the removal of stopwords, the model is less likely to overfit the training data and can capture more meaningful patterns that contribute to improved performance on the validation or test dataset.

I. Why is data transformation a good idea?

Data transformation is a valuable technique in the context of semantic-based databases for machine learning (ML) for several reasons:

1) Normalization and Standardization: By bringing the data into a consistent format and scale, data transformation techniques like normalization and standardization can enhance the performance of machine learning models.

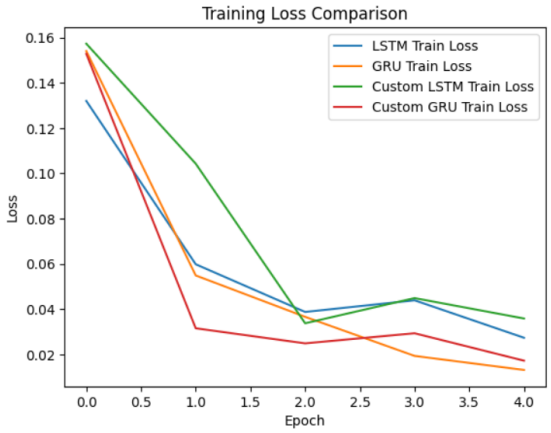

Fig. 20. Training Loss Comparison

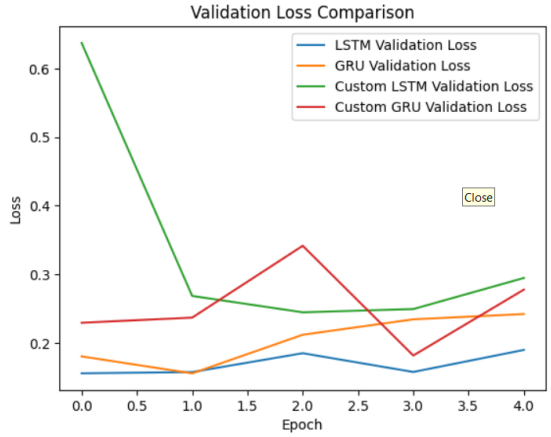

Fig. 21. Validation Loss Comparison

This guarantees that the machine learning algorithms evaluate features with varying sizes and units identically, resulting in more reliable and accurate model predictions.

2) Feature Engineering: To more accurately depict the underlying patterns in the data, new features can be created or old features can be modified through data transformation. TF-IDF (Term Frequency-Inverse Document Frequency) feature extraction approaches, for instance, can convert text input into numerical representations that are appropriate for machine learning algorithms.
3) Managing Non-Linearity: Although real-world data frequently displays non-linear correlations, many machine learning algorithms presume linearity in the data. By capturing non-linear patterns in the data, data transformation techniques like polynomial transformations and kernel approaches can improve the data's suitability for machine learning modeling.
4) Dimensionality Reduction: High-dimensional data in semantic databases may result in computational inefficiencies and overfitting of machine learning models. Principal component analysis (PCA) and t-SNE (t-distributed stochastic neighbor embedding) are two examples of data transformation techniques that can reduce the dimensionality of the data while maintaining its semantic structure, allowing for more effective and efficient machine learning modeling.
5) Handling Skewness and Outliers: Data with skew distributions and outliers can negatively impact ML model performance. By reducing the effects of skewness and outliers, data transformation methods like resilient scaling and log transformations can increase the stability and robustness of machine learning systems.

J. Why is LSTM and GRU the worst options?

While GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory) aren't always the worst possibilities, they might not be the greatest options in some situations, especially if computational efficiency is an issue or the data doesn't show long-term dependencies. For the following reasons, LSTM and GRU might not be the best options in some situations:

1) Complexity and Overfitting: To capture long-term dependencies in sequential data, LSTM and GRU are complicated models. However, utilizing LSTM or GRU may result in overfitting or needless complexity in the model if the data is rather basic or does not contain long-term dependencies.
2) Computational Efficiency: In comparison to more straightforward models such as feedforward neural networks or more straightforward recurrent neural network (RNN) architectures, LSTM and GRU models require a lot of processing. Using LSTM or GRU might not be feasible in situations when computational resources are scarce or efficiency is a top concern.
3) Data Size: In order for LSTM and GRU models to efficiently understand complicated patterns and long-term dependencies, they usually need a lot of training data. Training LSTM or GRU models with little data may result in subpar generalization performance or necessitate complex data augmentation methods.
4) Interpretability: As black-box models, LSTM and GRU models can be difficult to understand in terms of how they generate predictions. Simpler, more easily interpretable models may be preferred in applications where interpretability is critical, such as in certain medical or regulatory contexts.
5) Training Difficulty: Because of problems like vanishing gradients and bursting gradients, training LSTM and GRU models can be more difficult than training simpler models. Using simpler architectures might be more appropriate in situations where training stability is an issue or if the model needs to be trained rapidly.

K. How to handle larger number of text sizes while using GloVe? Why is that a bad idea?

Handling a larger number of text sizes while using GloVe embeddings can be challenging and may not be the most efficient approach for several reasons:

1) Memory Usage: GloVe embeddings typically require significant memory resources, especially when dealing with a large vocabulary and multiple text sizes. Storing and processing embeddings for a large number of text sizes can quickly become computationally expensive and memory-intensive, potentially leading to scalability issues.
2) Training Complexity: Training GloVe embeddings for multiple text sizes simultaneously can significantly increase the complexity of the training process. It may require custom modifications to the training algorithm and additional hyperparameter tuning to ensure convergence and optimal performance.
3) Model Overhead: Using GloVe embeddings for a large number of text sizes can introduce unnecessary model overhead, leading to increased model size and longer inference times. This can be particularly problematic in resource-constrained environments or applications where efficiency is a priority.
4) Limited Generalization: GloVe embeddings are trained on large corpora of text data and are designed to capture semantic relationships between words. However, embeddings trained on text data of varying sizes may not generalize well to unseen data or may capture spurious correlations specific to the training data, leading to reduced model performance.
5) Difficulty in Interpretation: Handling a larger number of text sizes with GloVe embeddings can make it more challenging to interpret and analyze the learned embeddings effectively. It may be harder to understand the underlying semantic relationships between words or to diagnose potential issues with the embedding space.

Instead of using GloVe embeddings for a larger number of text sizes, alternative approaches can be considered:

1) Embedding Size Selection: Choose a fixed embedding size that is appropriate for the majority of text sizes in the dataset. This simplifies the training process and reduces memory and computational requirements.
2) Text Preprocessing: Standardize text sizes through preprocessing techniques such as tokenization, padding, or truncation to ensure consistency before applying GloVe embeddings.
3) Fine-tuning Pretrained Embeddings: Instead of training GloVe embeddings from scratch, fine-tune pretrained embeddings on a specific task or domain-relevant data to adapt them to different text sizes.
4) Using Transform Tools: It eliminates the entire disadvantage of defining text sizes.

V. Conclusion

All models, including the ones implemented from scratch (Custom LSTM and Custom GRU), demonstrate strong performance in sentiment analysis tasks. The choice of model should be made based on considerations beyond performance alone, taking into account factors such as computational efficiency and interpretability. Based on the comprehensive evaluation of the models, it is evident that the LSTM library model outperformed the others, achieving the highest accuracy, precision, recall, and F1-score. This indicates that the LSTM library model effectively captured the underlying patterns in the data and made accurate predictions. On the other hand, the Custom GRU model demonstrated the lowest overall performance, with slightly lower accuracy and slightly lower precision, recall, and F1-score values compared to the other models. Therefore, if the primary goal is to maximize predictive performance and obtain the most accurate results, the LSTM library model would be the preferred choice. However, it's essential to consider factors such as computational efficiency, scalability, and ease of implementation when selecting the model for deployment in real-world applications. Additionally, further experimentation and fine-tuning may be necessary to optimize the performance of the models and address any specific requirements or challenges posed by the dataset or the task at hand.

Based on plotted values, The LSTM Library appears to be the most reliable choice among the algorithms evaluated, demonstrating better convergence, optimization, and generalization performance. The Custom LSTM Library may require further investigation to address issues related to training convergence and generalization to improve its performance.

The performance differences between custom implementations and library functions can be attributed to variations in optimization techniques, model architectures, regularization methods, and training procedures. While custom implementations offer flexibility and customization options, they may require more effort and expertise to achieve comparable performance to well-established library functions.

The decision to remove stopwords during text preprocessing should be made carefully, considering the specific characteristics of the dataset and the requirements of the classification task. While stopwords removal can help reduce noise and improve computational efficiency, it may also lead to overfitting in some cases, especially when the dataset is relatively small or when stopwords contain important semantic information. Experimentation and careful evaluation of different preprocessing strategies are essential to determine the optimal approach that maximizes model performance and generalization capabilities.

References

[1] Is violent content really against Reddit's rules?, 2023, Reddit
[2] Subreddit with videos of violence in public? No gore, just videos of fights and perilous situations in public settings, 2023, Reddit
[3] Update on Sitewide Rules Regarding Violent Content, 2018, Reddit
[4] Violent Comments on Reddit, April 2024, Reddit
[5] Do not post violent content, 2023, Reddit
[6] Simple Step-by-Step Guide On Reddit Sentiment Analysis, Riley Walz, January 2024
[7] Reddit Sentiment Analysis, by sprout social
[8] Reddit Sentiment Analysis Data Pipeline, by Arpit Nama, 2023
[9] Performing Sentiment Analysis On Your Favorite Subreddits, 2022, Reddit
[10] Sentiment Analysis on Reddit News Headlines with Python's Natural Language Toolkit (NLTK), by Brendan Martin and Nikos Koufos, 2023
[11] Enhancing machine learning-based sentiment analysis through feature extraction techniques, by Noura A. Semary, Wesam Ahmed, Khalid Amin, Paweł Pławiak, Mohamed Hammad, February 14, 2024

[12] Sentiment analysis using machine learning: Progress in the machine intelligence for data science, by G. Revathy a, Saleh A. Alghamdi b, Sultan M. Alahmari b, Saud R. Yonbawi c, Anil Kumar d, Mohd Anul Haq, October 2022
[13] Understanding sentiment analysis using machine learning, by Dovetail Editorial Team, Miroslav Damyanov, 16 August 2023
[14] What is sentiment analysis? Using NLP and ML to extract meaning, by Maria Korolov, September 2021
[15] Sentiment Analysis and Machine Learning, by Rachel Wolff, April 20, 2020
[16] What Is Open-Source Intelligence?, by Xcitium
[17] OSINT Techniques, by Neotas
[18] What is Open-Source Intelligence?, by Ritu Gill, February 23, 2023
[19] OSINT in Due Diligence: Minimizing Risk Through Open Data, April 2023
[20] What Is Open Source Intelligence: The Importance of OSINT in Your Organization's Threat Landscape, by Flashpoint
[21] Imbalanced data preprocessing techniques for machine learning: a systematic mapping study, by Vitor Werner de Vargas, Jorge Arthur Schneider Aranda, Ricardo dos Santos Costa, Paulo Ricardo da Silva Pereira and Jorge Luis Victória Barbosa, November 09, 2022
[22] Data Preprocessing in Machine Learning: 7 Easy Steps To Follow, by Kechit Goyal, February 18, 2024
[23] A Novel Machine Learning Data Preprocessing Method for Enhancing Classification Algorithms Performance, by Theodoros Iliou, Christos-Nikolaos Anagnostopoulos, Marina Nerantzaki, George Anastassopoulos, September 2015
[24] Data Preprocessing Techniques: 6 Steps to Clean Data in Machine Learning, by Nicolas Azevedo, September 29, 2023
[25] How to Choose Data Preparation Methods for Machine Learning, by Jason Brownlee, July 15, 2020
[26] How to Choose the Right Data for Your Computer Vision Project, by Akruti Acharya, February 13, 2023
[27] The Essential Guide to Quality Training Data for Machine Learning, by Jared P. Lander, Michael Beigelmacher, February 2020
[28] How to Choose the Right Machine Learning Algorithm: A Pragmatic Approach, by Iryna Sydorenko, May 3, 2021.
[29] Machine Learning Algorithms, by Geeks for Geeks, November 15, 2023
[30] The Top 10 Machine Learning Algorithms to Know, by James Le, March 05, 2024
[31] Top 10 Machine Learning Algorithms in 2022, by Vijay Kanade, May 30, 2022
[32] Top 10 Machine Learning Algorithms For Beginners: Supervised, and More, By Simon Tavasoli, February 9, 2024
[33] 10 Machine Learning Algorithms to Know in 2024, by Coursera Staff, April 1, 2024
[34] Illustrated Guide to LSTMs and GRUs: A step-by-step explanation, by Michael Phi, September 24, 2018
[35] 5 Types of LSTM Recurrent Neural Networks, by Exxact, December 28, 2023
[36] A Hierarchical Bidirectional GRU Model With Attention for EEG-Based Emotion Classification, by J. X. Chen, D. M. Jiang, AND Y. N. Zhang, September 2019
[37] Sentiment Analysis using RNN, LSTM, GRU, and Bi LSTM, by Saurabh Roy, 2020
[38] Sentiment Analysis with an Recurrent Neural Networks (RNN), by Geeks for Geeks, October 14, 2022
[39] Sampling-based algorithms for optimal motion planning, by Sertac Karaman and Emilio Frazzoli, June 2011
[40] Sentiment Analysis on Reddit Comments and Posts for OSINT Purposes, by Rahul Ravi Hulli