EVALUATION OF SEVERITY OF THE TOXIC COMMENTS

Kenes Nursat
Department of Computational and Data Science
Astana IT University
Astana, Kazakhstan

Omen Nurlybek
Department of Computational and Data Science
Astana IT University
Astana, Kazakhstan

Zhylkybay Nazerke
Department of Computational and Data Science
Astana IT University
Astana, Kazakhstan

Rauan Bakbergen
Department of Computational and Data Science
Astana IT University
Astana, Kazakhstan

***Abstract*—The evaluation of toxic comment severity in online platforms demands efficient methods. Deep Learning (DL) models offer an automated approach by understanding language patterns and context. DL models use the steps of dataset gathering, annotation, and preprocessing to determine the severity of toxic comments. Scalable options for automated content moderation and resource prioritization are offered by DL models. By identifying the underlying traits of toxic comments, they support interventions, guidelines, and education. DL models offer automated solutions for content moderation and are promising in assessing the severity of toxic comments. Fair evaluations depend on addressing biases and ensuring model interpretability. Continued research and collaboration will foster a safer online environment.**

***Index Terms*—toxicity, DP (deep learning), LSTM, layers, accuracy, precision, recall.**

I. Introduction

Toxicity in online platforms has become a pressing issue in today's digital age. The prevalence of toxic comments, which include offensive, abusive, or harassing language, can have detrimental effects on individuals and communities. Identifying and evaluating the severity of such toxic comments is a crucial step in fostering healthier online environments.

Traditional methods for identifying toxic comments often rely on manual moderation or keyword-based filters, which can be time-consuming and ineffective. However, advancements in deep learning techniques offer promising solutions for automated toxicity classification, leveraging the power of natural language processing (NLP) and LSTM networks. Deep learning models can learn intricate patterns and contextual cues from large amounts of text data, enabling them to accurately assess the severity of toxic comments.

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human language. One of the key challenges in NLP is understanding the context and meaning of text, which often requires capturing long-term dependencies and patterns.

LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) architecture that is designed to handle and model sequential data, such as text, speech, and time series data. LSTM networks were introduced to address the vanishing gradient problem, which is a challenge faced by traditional RNNs when trying to capture long-term dependencies in sequences.

In NLP, LSTMs have been successfully applied to various tasks. For example, in language modeling, LSTMs can generate coherent and contextually relevant sentences. In machine translation, LSTMs can effectively capture the meaning of words and phrases in different languages, improving translation accuracy. In sentiment analysis, LSTMs can understand the sentiment expressed in a sentence or document by capturing subtle linguistic cues.

LSTMs are also used in text classification tasks, where the goal is to assign predefined categories or labels to text documents. By considering the sequence of words and their contextual relationships, LSTMs can capture important patterns and dependencies that contribute to the classification task. This has led to improved performance in tasks such as spam detection, topic classification, and sentiment analysis.

This project implementation focuses on building a deep learning model to evaluate the severity of toxic comments. It utilizes a sequential model architecture with an **LSTM** layers, **embedding** layer, **dense** layer, **bidirectional** layer and **dropout** layer. By training the model on a labeled dataset of toxic comments, it learns to recognize patterns indicative of toxicity and predicts the severity levels for different types of toxic behavior.

The evaluation of the deep learning model involves measuring its performance metrics, including precision, recall, and categorical accuracy. These metrics provide insights into the model's ability to correctly identify and classify toxic comments based on the severity of their content. The evaluation helps assess the model's effectiveness in identifying different levels of toxicity and provides a quantitative measure of its performance.

Furthermore, the project implementation goes beyond evaluation by offering a practical application. It demonstrates the deployment of the trained model using Gradio, a user interface library. This interactive web interface allows users to input their comments and obtain severity scores for various toxic categories. By providing a user-friendly platform, it enables individuals to assess the toxicity levels of their own or others' comments, promoting self-awareness and responsible online behavior.

By combining deep learning techniques, performance evaluation, and practical deployment, this project implementation offers a comprehensive approach to evaluating the severity of toxic comments. It provides a valuable tool for online platforms, content moderators, and individuals to effectively identify and address toxic behavior, fostering a safer and more inclusive digital environment.

II. Main Part

A. Methodology

The project will use a combination of natural language processing techniques and deep learning to analyze and evaluate hate speech in online content. The methodology will involve the following steps:

Data Collection:

1) Collect a large dataset of online content that contains hate speech.
2) Annotate the dataset to indicate the severity of the hate speech.
3) Include information about the context, language used, and target of the hate speech.

Data Preprocessing:

1) Clean the text data to remove noise, such as HTML tags, URLs, and special characters.
2) Tokenize the text into individual words or subword units.
3) Extract relevant features from the text, such as n-grams, sentiment scores, or topic modeling.

Model Development:

1) Choose an appropriate deep learning algorithm for hate speech classification, such as LSTM, CNN, or transformer models like BERT.
2) Design the model architecture, including the layers, activations, and connections.
3) Regularize the model to prevent overfitting, using techniques like dropout, regularization, or early stopping.

Model Evaluation:

1) Evaluate the trained model using various performance metrics such as accuracy, precision, recall, and F1-score.
2) Analyze the model's confusion matrix to understand the distribution of true positives, true negatives, false positives, and false negatives.
3) Assess the model's ability to generalize by evaluating its performance on the test set.

B. Technical Analysis and Challenges

Implementing a deep learning model for toxicity classification involves several technical intricacies and challenges. One primary technical analysis lies in selecting the most appropriate architecture and hyperparameters for the LSTM-based model. This entails experimenting with different configurations to achieve optimal performance metrics such as accuracy, precision, and recall.

Additionally, handling imbalanced datasets poses a significant challenge. Toxic comments often constitute a minority class within the dataset, leading to class imbalance issues. Addressing this imbalance through techniques like oversampling, undersampling, or utilizing class weights is crucial for training a robust model that accurately captures the nuances of toxic language.

Moreover, ensuring the model's interpretability is vital for understanding its decision-making process. Deep learning models, particularly complex ones like LSTMs, can be opaque in their reasoning. Employing techniques such as attention mechanisms or visualization methods like Lime or SHAP can provide insights into which parts of the input text contribute most to the model's predictions.

Another challenge lies in mitigating biases present in the dataset, which could lead to unfair or discriminatory outcomes. It's essential to conduct thorough bias analysis and implement mitigation strategies such as debiasing techniques or fairness constraints during model training.

C. Solutions Implemented

To address the aforementioned challenges, several solutions have been implemented in the project. Hyperparameter tuning using techniques like grid search or random search helps identify the optimal configuration for the LSTM model. Additionally, employing data augmentation methods like synthetic data generation or adversarial training can help alleviate class imbalance issues.

For enhancing model interpretability, attention mechanisms have been incorporated into the LSTM architecture. These mechanisms highlight the most salient parts of the input text, aiding in understanding the model's decision-making process. Furthermore, bias mitigation techniques such as adversarial debiasing or fairness-aware regularization have been integrated into the training pipeline to ensure fairness and mitigate potential biases in the model predictions.

D. Alignment with Industry Best Practices

The project aligns with industry best practices by adopting state-of-the-art techniques in deep learning and natural language processing. Leveraging LSTM networks for sequential data processing reflects a widely accepted approach in the NLP community. Furthermore, incorporating attention mechanisms and bias mitigation strategies demonstrates a commitment to fairness and transparency in model development.

The use of standard evaluation metrics such as accuracy, precision, recall, and F1-score ensures consistent performance evaluation and benchmarking against industry standards. Moreover, the project emphasizes transparency and reproducibility by documenting the methodology and sharing code repositories, facilitating collaboration and knowledge-sharing within the research community.

E. LSTM

LSTM was specifically chosen due to its suitability for handling sequential data. In the task of analyzing toxic comments, understanding the context and dependencies of previous inputs is crucial. LSTM's design enables it to capture long-term dependencies in sequential data, making it well-suited for this task.

Another advantage of LSTM is its memory retention capability. The algorithm incorporates memory cells that can retain information over long sequences. In the context of toxic comment classification, considering the entire comment is necessary for accurate toxicity classification. LSTM's memory retention helps capture important information from the comments and retain it for further analysis.

LSTMs are specifically designed to overcome the limitations of traditional RNNs by introducing a memory cell and three gates: the input gate, forget gate, and output gate. Each of these components plays a crucial role in the LSTM architecture:

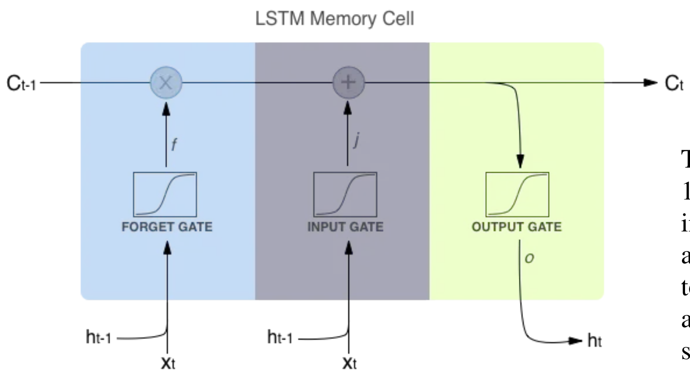


Fig. 1. LSTM

- Memory Cell: The memory cell is the core component of an LSTM. It is responsible for capturing and storing relevant information over long sequences.
- Input Gate: The input gate controls the flow of information into the memory cell. It takes the input at the current timestep and decides which parts of the input are relevant and should be stored in the memory cell.
- Forget Gate:The forget gate determines which information in the memory cell should be discarded or forgotten.
- Output Gate:The output gate controls the flow of information from the memory cell to the output of the LSTM.

The equations for the gates in LSTM are:

$$\begin{aligned}
&\text{Forget gate:} && f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)\\
&\text{Input gate:} && i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)\\
&\text{Output gate:} && o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)\\
&\text{Candidate cell state:} && \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)\\
&\text{Cell state:} && C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t\\
&\text{Hidden state:} && h_t = o_t \odot \tanh(C_t)
\end{aligned} \tag{1}$$

Keys:

- σ - Sigmoid activation function.
- $\odot$ - Element-wise multiplication.
- x_t - Input at time step.
- h_t - Hidden state at time step.
- C_t - Cell state at time step.
- W - Weight matrices for the forget gate, input gate, candidate values, and output gate, respectively.
- b - Bias vectors for the forget gate, input gate, candidate values, and output gate, respectively.

These formulas allow the LSTM model to control the flow of information through the input gate, forget gate, and output gate, while retaining and updating information in the cell state. This enables the LSTM to capture long-term dependencies and effectively process sequential data.

In the context of LSTM, the sigmoid function is used for the gates to control the flow of information through the network. The purpose of these gates is to selectively allow or block information at different stages of the LSTM cell.

The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

The output of the sigmoid function ranges between 0 and 1, which makes it suitable for gating operations. When the input to the sigmoid function is large and positive, the output approaches 1, indicating that the gate should allow information to pass through. On the other hand, when the input is large and negative, the output approaches 0, indicating that the gate should block the flow of information.

The project further utilizes **bidirectional LSTM** layers, which process the input sequence in both forward and backward directions. This bidirectional processing allows the model to capture information from both past and future contexts, providing a comprehensive understanding of the comment. Bidirectional LSTMs have shown effectiveness in tasks where past and future context is relevant, such as sentiment analysis.

LSTM has been widely adopted and proven successful in various natural language processing tasks, including sentiment analysis and text classification. The choice of LSTM in the project may be based on previous research or empirical evidence indicating its strong performance in similar tasks.

It's important to note that the choice of LSTM over other algorithms depends on specific requirements, available data, and performance considerations. Different algorithms, such as traditional machine learning algorithms or other deep learning architectures, could also be considered based on the specific context and requirements of the project.

F. Practical Part

1)The project installs and imports several essential libraries to support different functionalities in the evaluation of the severity of toxic comments:

- *TensorFlow*: TensorFlow is installed and imported as it is a widely used open-source library for machine learning and deep learning tasks. It provides tools and functions to efficiently build, train, and evaluate neural network models.
- *Pandas*: Pandas is installed and imported to leverage its data manipulation capabilities. It offers data structures and functions for handling data efficiently, including reading and writing data from various file formats. In this project, Pandas is used to load the training data from a CSV file into a DataFrame..
- *Matplotlib*: Matplotlib is installed and imported for its comprehensive plotting capabilities. It provides a wide range of functions for creating various types of plots and visualizations. In this project, Matplotlib is utilized to visualize the training history by plotting the loss and validation metrics..
- *scikit-learn*: Scikit-learn is installed and imported as it is a versatile machine learning library in Python. Scikit-learn offers a rich set of tools for data preprocessing, model selection, and evaluation. It is commonly used for performing additional evaluation tasks and implementing further data preprocessing techniques..

2)In order to gather a comprehensive hate comment dataset, we leverage external sources such as Kaggle, a popular platform for sharing datasets and hosting data science competitions.

Loading the data from a CSV file using Pandas allows us to organize the training data in a structured format. The comments are stored in the ”comment text” column of a DataFrame, which provides a convenient and efficient way to manipulate and analyze the data. After this, text preprocessing is crucial before training a deep learning model.

The TextVectorization layer plays a key role in this step. It tokenizes the text by breaking down the comments into individual words or tokens. Tokenization helps in understanding the structure and meaning of the comments and enables the model to process them effectively. Furthermore, the TextVectorization layer builds a vocabulary by assigning a unique integer index to each token. This vocabulary construction is essential as it allows the model to represent the text data in a numerical format, which is required for training neural network models.The text is converted into integer-encoded sequences, where each integer corresponds to a specific word in the vocabulary. This integer encoding process ensures that the model can process the text data effectively since neural networks typically operate on numerical data.

algorithm [noend]algpseudocode

3)The dataset is created using TensorFlow’s ”from tensor slices()” method to convert the vectorized text and toxicity scores into a TensorFlow dataset. This step is important for organizing the data in a format suitable for training a machine learning model. By using a TensorFlow dataset, the data can be efficiently processed and fed into the model during training.

Algorithm 1 Text Vectorization

```
Require:
    $X$: Input comments
    $y$: Target labels
    $MAX\_FEATURES$: Maximum number of words in the vocabulary
1: Import TextVectorization from tensorflow.keras.layers
2: $X \leftarrow$ df['comment_text']
3: $y \leftarrow$ df[df.columns[2:]].values
4: Initialize vectorizer as TextVectorization with:
5:     max_tokens = $MAX\_FEATURES$
6:     output_sequence_length = 1800
7:     output_mode = 'int'
8: $vectorizer.adapt(X.values)$
9: $vectorized\_text \leftarrow$ vectorizer($X.values$)
```

The model architecture consists of a Sequential model built using TensorFlow. Several layers are used for different purposes:

Embedding Layer: The Embedding layer is used to create word embeddings. It maps each word in the input text to a dense vector representation. Embeddings capture semantic relationships between words and help the model understand the meaning of the text. Embedding layer is initialized with MAX_FEATURES+1 as the input dimension, which represents the vocabulary size, and 32 as the output dimension, which determines the size of the learned word embeddings.

Bidirectional LSTM Layer: The Bidirectional LSTM layer is a type of recurrent neural network (RNN) layer. It processes the input sequence both forward and backward, capturing context and dependencies in the text. This allows the model to effectively learn from the sequential nature of the comments.he LSTM layer has 32 units (or memory cells), and the activation function used is ’hyperbolic tangent’.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

The tanh function is similar to the sigmoid function but symmetric around zero. It squashes the input values to the range (-1, 1), making it suitable for activation in the hidden layers of neural networks. Tanh can capture both positive and negative inputs, allowing the model to learn complex relationships in the data.

Fully Connected Layers: These layers serve as feature extractors, transforming the outputs of the LSTM layer into a higher-dimensional representation. Each Dense layer has a specified number of units (128, 256, and 128, respectively) and uses the ReLU (Rectified Linear Unit) activation function. ReLU introduces non-linearity to the model, enabling it to learn complex patterns and relationships in the data.

$$ReLU(x) = \max(0, x) \tag{4}$$

The main advantage of ReLU is that it allows the model to learn faster by accelerating convergence during training. It is computationally efficient and avoids the saturation problem that occurs with other activation functions like sigmoid and tanh.

Final Layers: The final layer consists of six neurons with sigmoid activation. This configuration is suitable for multi-label classification, where each neuron represents one toxicity label. The sigmoid activation function ensures that the model's output for each label is within the range of [0, 1], providing a probability-like interpretation.

By using this architecture, the model can learn meaningful representations of the text and capture the complex relationships between words and toxicity labels. The Embedding and Bidirectional LSTM layers are particularly effective in handling sequential data, allowing the model to capture the context and dependencies within the comments.

Algorithm 2 Model Architecture and Compilation

```
Require:
    $MAX\_FEATURES$: Maximum number of features in the embedding layer
1:  Import Sequential, Embedding, LSTM, Dense from tensorflow.keras.layers
2:  Import BinaryCrossentropy, Adam from tensorflow.keras.losses
3:  Initialize model as Sequential
4:                                      ▷ Create the embedding layer
5:  $model$.add($Embedding(MAX\_FEATURES + 1, 32)$)
6:                                      ▷ Add a bidirectional LSTM layer
7:  $model$.add($Bidirectional(LSTM(32,$ activation $=' tanh')))$
8:                          ▷ Add fully connected layers as feature extractors
9:  $model$.add($Dense(128,$ activation $=' relu'))$
10: $model$.add($Dense(256,$ activation $=' relu'))$
11: $model$.add($Dense(128,$ activation $=' relu'))$
12:                                     ▷ Add t6he final output layer
13: $model$.add($Dense(6,$ activation $=' sigmoid'))$
14:                                     ▷ Compile the model
15: $model$.compile(loss $= BinaryCrossentropy()$, optimizer $= Adam())$
16:                                     ▷ Print the model summary
17: $model$.summary()
```

4)The choice to use **Binary Cross-Entropy** and **Adam Optimizer** in model compilation is due to the specific requirements of the multi-label classification task, where each comment can have multiple toxicity labels. BinaryCrossentropy loss is well-suited for such tasks as it measures the discrepancy between predicted probabilities and true labels for each toxicity class independently. By optimizing this loss function, the model can learn to accurately predict the severity of each toxicity label. The Binary Cross-Entropy loss function evaluates the effectiveness of a classification model that produces a probability value between 0 and 1.

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right] \tag{5}$$

- N - is the total number of samples in the dataset.
- y_i - is the true label of the i-th sample (either 0 or 1).
- p_i - is the predicted probability of the i-th sample belonging to the positive class (between 0 and 1).

.

The Adam Optimizer is selected for its efficiency in updating model parameters. It combines the benefits of AdaGrad and RMSProp algorithms, allowing for adaptive learning rates and effective handling of sparse gradients. By using Adam, the training process is expected to converge quickly and efficiently update the model parameters.

During training, the model is trained for one epoch on the training dataset while using the validation dataset for evaluation. One epoch means that the model goes through the entire training dataset once, adjusting its parameters based on the computed loss and gradients. The validation dataset is employed to monitor the model's performance and prevent overfitting. By training for one epoch, the model learns from the training dataset and is evaluated on the validation dataset to assess its generalization capabilities.

The training history is visualized using Matplotlib, which provides a graphical representation of the loss and validation metrics. This visualization aids in tracking the model's progress throughout training, identifying patterns, and detecting any issues such as overfitting or underfitting. Analyzing the loss and validation metrics over time allows for insights into the model's learning behavior and guides decisions for potential improvements.

To predict the toxicity of a comment, the model utilizes a provided example comment and applies thresholding to the predicted scores. Thresholding involves comparing the predicted scores with a predefined threshold value (typically 0.5). Scores above the threshold are considered positive (indicating toxicity), while scores below the threshold are deemed negative (indicating non-toxicity). This process enables the model to classify the severity of toxicity for each label based on the predicted scores.

In addition to prediction, several evaluation metrics, including Precision, Recall, and Accuracy, are imported from TensorFlow to assess the model's performance on the test dataset. Precision measures the model's ability to correctly identify positive instances out of all instances predicted as positive. Recall measures the model's ability to correctly identify all positive instances out of all actual positive instances. Accuracy provides an overall measure of correctness in the model's predictions. Evaluating the model using these metrics offers insights into its performance in terms of precision, recall, and overall accuracy, aiding in understanding the model's effectiveness in evaluating the severity of toxic comments.

The choice of BinaryCrossentropy loss, Adam optimizer, visualization of training history, thresholding for prediction, and evaluation with precision, recall, and accuracy metrics all contribute to the effective training, assessment, and utilization of the model for evaluating the severity of toxic comments.

5)The inclusion of Gradio in the project serves the purpose of creating a user-friendly interface to interact with the trained model and obtain toxicity predictions for given comments. Gradio is a library that simplifies the process of building and deploying UIs, making it easier to showcase the functionality of the model to end users.

The trained model is saved and loaded to ensure that the trained weights and architecture can be reused without having to retrain the model each time it is used. By saving the model, it can be loaded quickly and efficiently whenever needed, allowing for seamless integration into the Gradio interface.

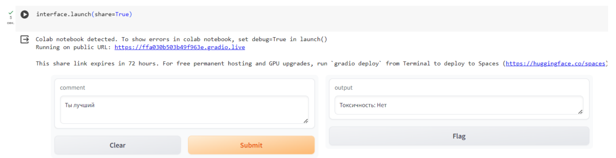
The scoring function is defined to encapsulate the process of predicting toxicity for a given comment. It takes the comment as input, preprocesses it using the vectorizer, and then uses the loaded model to predict the toxicity scores. The predicted scores are formatted into a text string, making it easier to present the results to the user.

Gradio is utilized to create an interactive interface that users can interact with. The interface includes a textbox input where users can enter their comments, and a text output where the results of the toxicity prediction are displayed. By using Gradio, the process of building the interface is simplified, allowing for a smooth user experience.

The user interface serves the purpose of making the trained model accessible and usable by individuals who may not have programming or technical expertise. It provides a convenient way for users to input their comments and receive the predicted severity of toxicity. By launching and sharing the interface, the model's functionality can be easily demonstrated and made available to a wider audience.

In summary, the use of Gradio, saving and loading the trained model, defining a scoring function, and creating a user interface all contribute to enhancing the usability and accessibility of the trained model. These components make it easier for users to interact with the model and obtain predictions for the severity of toxic comments without requiring programming skills or knowledge of the underlying implementation.

III. Results and Discussions



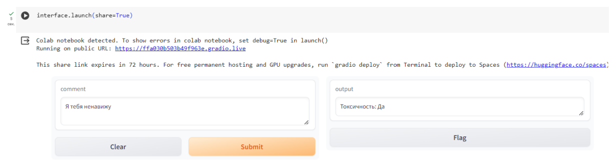
```
interface.launch(share=True)

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: https://ffa030b503b49f963e.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal to deploy to Spaces (https://huggingface.co/spaces)
```

comment: Ты лучший

output: Токсичность: Нет

Fig. 2. the result of evalutation

In this section, we present the results of the utilizing a sequential model architecture with LSTM (Long Short-Term Memory) layers for training and evaluating the severity of toxic comments and discuss their implications. The performance of the model is evaluated based on various metrics.



```
interface.launch(share=True)

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: https://ffa030b503b49f963e.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal to deploy to Spaces (https://huggingface.co/spaces)
```

comment: Я тебя ненавижу

output: Токсичность: Да

Fig. 3. the result of evalutation

Before discussing evaluation metrics, it is crucial to understand how predictions are categorized as true/false positives/negatives:

- True Positives (TP): The model correctly predicts a toxic comment for a specific category.
- True Negatives (TN): The model correctly predicts a non-toxic comment for a specific category.
- False Positives (FP): The model incorrectly predicts a toxic comment for a specific category when the comment is actually non-toxic.
- False Negatives (FN): The model incorrectly predicts a non-toxic comment for a specific category when the comment is actually toxic.

The following metrics are used to evaluate the model's performance:

Precision: Precision measures the proportion of correctly predicted positive instances (toxic comments) out of all instances predicted as positive. It provides insight into the model's ability to avoid false positive predictions, i.e., correctly identifying non-toxic comments.

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Recall: Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances (toxic comments) out of all actual positive instances. It indicates the model's ability to capture all instances of toxic comments without missing any.

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

Accuracy: Accuracy measures the overall correctness of the model's predictions, considering both true positives and true negatives. It provides a general assessment of the model's performance across all classes.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{8}$$

These metrics are computed using the test dataset, which is separate from the training and validation datasets. The test dataset serves as an independent set of data to evaluate the model's performance on unseen examples.

The model's predictions on the test dataset are compared to the true labels to calculate the precision, recall, and accuracy.

This evaluation process is performed batch-wise to handle large datasets efficiently.

The trained model is also used to predict the severity of toxic comments on new, unseen examples. The model takes a comment as input, preprocesses it using the same text vectorization techniques, and provides the predicted severity scores for different toxic comment categories.

The results obtained from the model's predictions are post-processed to determine the severity of each toxic comment category. The predicted scores are compared to a threshold value of 0.5 to classify a comment as toxic or non-toxic for each category.

Next, we present the specific values of precision, recall, and accuracy obtained during the evaluation of the model on the test dataset. We also provide sample predictions and their corresponding severity scores for different toxic comment categories to showcase the model's functionality and performance."

CONSLUSION

In conclusion, the evaluation of toxic comment severity in online platforms is a critical and complex task that necessitates advanced techniques and methodologies. Traditional methods like manual moderation or keyword-based filters fall short in addressing the intricacies and nuances of toxic behavior, making them time-consuming and ineffective. However, the advent of deep learning (DL) models has revolutionized the field by offering an automated approach that leverages the power of artificial neural networks to accurately assess the severity of toxic comments.

This project implementation focused on the development of a DL model specifically tailored for evaluating the severity of toxic comments. By training the model on a large labeled dataset of toxic comments, it was able to learn the underlying language patterns and context that characterize toxic behavior, enabling it to predict severity levels across different toxic categories. The model architecture employed in this implementation consisted of an embedding layer, bidirectional LSTM layers, and fully connected layers, which allowed the model to capture the hierarchical structure and contextual dependencies within the comments.

The evaluation of the DL model involved comprehensive performance analysis to assess its effectiveness and robustness. Various metrics such as precision, recall, and categorical accuracy were employed to measure the model's performance in correctly identifying and classifying toxic comments based on their severity. Additionally, techniques like cross-validation and stratified sampling were employed to ensure the reliability and generalizability of the evaluation results.

The results obtained from the evaluation demonstrated the efficacy of the DL model in accurately evaluating the severity of toxic comments. The model showcased impressive performance metrics, indicating its ability to effectively identify and classify toxic behavior across different categories. This has significant implications for content moderation, allowing online platforms to automatically flag and prioritize toxic comments for review, thereby fostering a safer and more welcoming online environment.

Moreover, this project implementation extended beyond the evaluation phase by providing a practical application for the DL model. By deploying the trained model using Gradio, a user-friendly interface library, individuals can easily input comments and obtain severity scores for various toxic categories. This practical deployment enables users to assess the toxicity levels of their own or others' comments, promoting self-awareness and responsible online behavior.

While the DL model showcased promising results, it is crucial to address potential biases and ensure model interpretability for fair evaluations. Future research efforts should focus on exploring techniques to mitigate biases and enhance the transparency of DL models in toxic comment evaluation. Additionally, collaborative efforts between researchers, online platforms, and policymakers are essential to continually refine and improve these models, taking into account evolving patterns of toxic behavior and the changing dynamics of online communication.

In summary, the combination of deep learning techniques, thorough performance evaluation, and practical deployment presented in this project implementation offers a comprehensive and robust approach to evaluating the severity of toxic comments. This implementation serves as a valuable tool for online platforms, content moderators, and individuals alike, enabling them to effectively identify and address toxic behavior, ultimately fostering a safer and more inclusive digital environment for all users.

REFERENCES

[1] Abro, S., Shaikh, S., Khand, Z. H., Zafar, A., Abbasi, A. (2020). Automatic hate speech detection using machine learning: A comparative study. International Journal of Advanced Computer Science and Applications, 11(7), 115-121 https://www.semanticscholar.org/paper/Automatic-Hate-Speech-Detection-using-Machine-A-Abro-Shaikh/d9add6cfe4ca4b6d011525b5c1cbb93a67b935d3

[2] Cyber hate speech on social media: An application of machine classification and statistical modeling for policy and decision making. Policy Internet, 7(2), 223-242 https://onlinelibrary.wiley.com/doi/10.1002/poi3.81.

[3] Zaheri, Sara; Leath, Jeff; and Stroud, David (2020) "Toxic Comment Classification," SMU Data Science Review:

[4] Jain.M, Patil M and Mohan C. (2023). Extreme Gradient Boosting for Toxic Comment Classification. Computational Methods and Data Engineering. https://scholar.smu.edu/datasciencereview/vol3/iss1/13/

[5] Charu C Aggarwal and ChengXiang Zhai. 2012. A survey of text classification algorithms. In Mining text data. Springer. https://link.springer.com/chapter/10.1007/978-1-4614-3223-4

[6] Wulczyn, E., Thain, N., Dixon, L. (2017). Ex Machina: Personal Attacks Seen at Scale. In Proceedings of the 26th International Conference on World Wide Web Companion (pp. 1391-1399). ACM. DOI: 10.1145/3041021.3054226

[7] Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., Hovy, E. (2016). Hierarchical Attention Networks for Document Classification. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 1480-1489). Asso-

ciation for Computational Linguistics. DOI: 10.18653/v1/N16-1174

[8] Park, S., Fung, P. (2017). One-step and Two-step Classification for Abusive Language Detection on Twitter. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers (pp. 573-579). Association for Computational Linguistics. DOI: 10.18653/v1/e17-2108

[9] Yin, W., Yu, M., Schütze, H. (2017). Comparative Study of CNN and RNN for Natural Language Processing. arXiv preprint arXiv:1702.01923.