Toxic Comment Classification Using Bidirectional LSTM with K-Fold Cross-Validation

Bryant Michelle Sarabia Ortega
ML4SE Course
University of L'Aquila
Email: bryantmichelle.sarabiaortega@student.univaq.it

***Abstract*—This paper presents a deep learning approach to toxic comment classification using a Bidirectional Long Short-Term Memory (BiLSTM) neural network for multi-label classification of Wikipedia comments. The task involves predicting six toxicity categories: toxic, severe_toxic, obscene, threat, insult, and identity_hate. We employ 10-fold cross-validation on the complete Jigsaw dataset (159,571 samples) to ensure robust evaluation. The implementation follows software engineering best practices, utilizing the Abstract Factory and Strategy design patterns to achieve flexible data persistence supporting both file-based and in-memory storage with minimal code changes. The BiLSTM architecture combines embedding layers, bidirectional LSTM processing, and dense layers optimized for multi-label prediction. Results demonstrate strong performance with mean F1-score of 0.6877 and low variance across folds, validating both the model's generalization capability and the architectural design's effectiveness.**

***Index Terms*—toxic comment classification, multi-label classification, natural language processing, deep learning, LSTM, text classification, design patterns, software architecture**

I. Introduction

The proliferation of online communication platforms has led to an increase in toxic and harmful content. Identifying and moderating such content is crucial for maintaining healthy online communities. This project addresses the challenge of automatically detecting toxic comments using machine learning approaches.

The Conversation AI team, a research initiative founded by Jigsaw and Google, developed the Perspective API to help identify toxic comments. This work builds upon their efforts by implementing and comparing various classification techniques on the Jigsaw Toxic Comment Classification Challenge dataset [1].

This paper is organized as follows: Section II describes the dataset and preprocessing methodology, Section III presents the software design patterns and architecture, Section IV presents the machine learning techniques employed, Section V explains the cross-validation approach, Section VI discusses the experimental results, and Section VII concludes the work.

II. Dataset Description

A. Multi-Label Classification

Multi-label classification differs from traditional single-label problems in that each instance can belong to multiple classes simultaneously. In our case, a comment can exhibit multiple types of toxicity. For example, a comment might be both *toxic* and *insult*, or *obscene*, *insult*, and *identity_hate* at the same time.

Formally, given an instance x and a set of labels $L = \{l_1, l_2, ..., l_k\}$, a multi-label classifier learns a function $h : X \to 2^L$ that maps instances to subsets of labels.

B. Dataset Analysis

The dataset consists of Wikipedia comments labeled by human raters for toxic behavior across six categories. Comments marked as toxic display general negativity or hostility, while severe toxic comments contain highly aggressive or harmful content surpassing typical toxic behavior. Obscene comments feature offensive or vulgar language, and threat comments convey explicit threats or intentions of harm towards individuals or groups. Insult comments contain disrespectful or demeaning language directed at others, whereas identity hate comments display discrimination or prejudice based on identity factors such as race, gender, or religion.

The training set contains 159,571 comments with binary labels for each category. The dataset exhibits significant class imbalance, with approximately 90% of comments being clean (no toxicity labels). Among the toxic categories, threat is the rarest with only 0.30% representation, while toxic is most common at 9.58% of all comments.

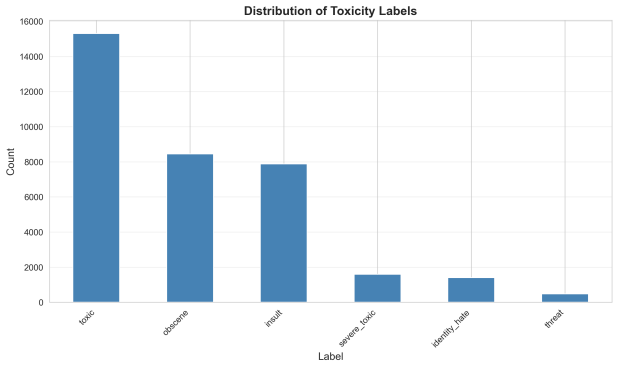

Fig. 1. Distribution of toxicity labels showing severe class imbalance

Figure 1 shows the class imbalance, while Figure 2 reveals that most toxic comments have only one label, with few having multiple toxicity types.

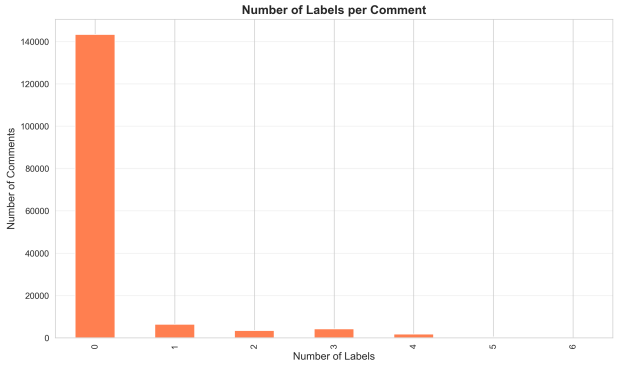

Fig. 2. Distribution of number of labels per comment

Fig. 3. Correlation matrix showing relationships between toxicity labels

C. Data Characteristics

Figure 3 shows strong correlations between certain label pairs. For instance, *toxic* correlates strongly with *obscene* and *insult*, suggesting these toxicity types often co-occur.

D. Text Length Analysis

Understanding the distribution of comment lengths provides insights into the dataset characteristics. Figure 4 shows the distribution of comment lengths in characters, revealing that most comments are relatively short, with a right-skewed distribution. Figure 5 presents a detailed analysis comparing comment lengths across different toxicity categories, demonstrating that toxic comments tend to have similar length distributions to clean comments, indicating that length alone is not a strong predictor of toxicity.

E. Preprocessing

Text preprocessing is critical for NLP tasks. Our pipeline begins with lowercasing all text and expanding contractions such as converting ”can’t” to ”cannot”. Non-alphabetic characters

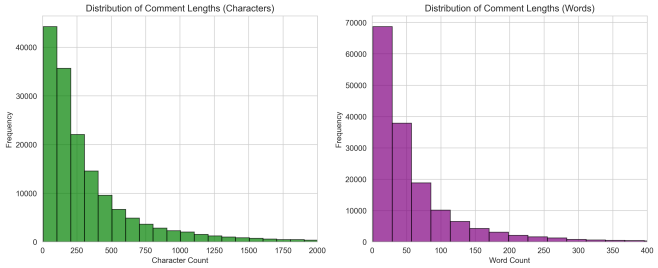

Fig. 4. Distribution of comment lengths showing right-skewed pattern

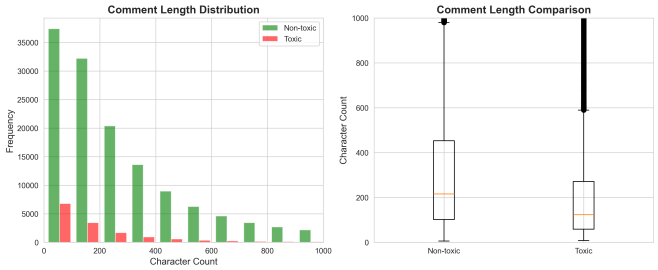

Fig. 5. Comment length analysis across toxicity categories

are removed, followed by tokenization of the text. We apply lemmatization using spaCy and remove common stopwords to focus on meaningful content.

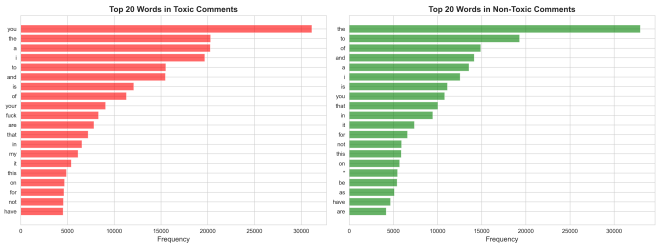

Fig. 6. Most frequent words after preprocessing

III. Software Design and Architecture

To satisfy the project requirements for flexibility and modularity, the application employs several key design patterns that enable seamless switching between different persistence layers while maintaining business logic integrity.

A. Abstract Factory Pattern

The data persistence layer implements the Abstract Factory design pattern through the `DataLoader` abstract base class. This pattern defines a common interface for data loading operations regardless of the underlying storage mechanism. The `DataLoader` class declares three abstract methods: `load_train_data()` for retrieving training samples and labels, `load_test_data()` for loading test samples, and `save_predictions()` for storing model outputs. Two concrete implementations fulfill this interface: `FileBasedDataLoader` reads data from CSV files in the filesystem, while `MemoryBasedDataLoader` operates on

pandas DataFrames already loaded in memory. A factory function `get_data_loader(mode, **kwargs)` instantiates the appropriate loader based on the specified mode parameter, returning either a file-based or memory-based implementation without exposing implementation details to the caller.

B. Strategy Pattern

The model architecture follows the Strategy pattern through the `BaseModel` abstract class, which defines a uniform interface for all classification models. This abstraction declares methods for training (`fit()`), prediction (`predict()`), and persistence (`save_model()`, `load_model()`), along with metadata accessors. The `BiLSTMModel` class implements this interface, encapsulating the specific deep learning architecture while exposing the same contract as any other model implementation. This design allows the application to treat different model types uniformly, facilitating experimentation with multiple approaches without modifying client code.

C. Separation of Concerns

The application architecture strictly separates data access, preprocessing, model training, and evaluation into distinct modules. The `src.data` package handles all data loading and preprocessing operations, while `src.models` contains model implementations, and `src.evaluation` manages metrics calculation and visualization. This modular organization ensures that changes to one layer do not cascade to others. For instance, switching from file-based to memory-based data loading requires only changing the factory function parameter from `mode="file"` to `mode="memory"`, with no modifications needed to preprocessing, training, or evaluation code.

D. Dependency Injection

The training pipeline receives data loaders, preprocessors, and models as constructor parameters rather than instantiating them directly. This dependency injection approach decouples component instantiation from usage, enabling easy substitution of implementations. The notebook demonstrates this pattern by creating a `FileBasedDataLoader` instance and passing it to subsequent operations, allowing straightforward replacement with a `MemoryBasedDataLoader` for testing or different deployment scenarios.

E. Implementation Benefits

This architectural design satisfies the project's non-functional requirements by providing true flexibility between persistence mechanisms. The business logic interacts exclusively with abstract interfaces, remaining agnostic to whether data resides in files or memory. Switching persistence layers involves changing a single line of code in the data loader instantiation, with the identical `DataLoader` interface guaranteeing that all downstream operations function unchanged. The modular structure also facilitates testing, maintenance, and future extensions such as database-backed storage or cloud-based data sources.

IV. Deep Learning Architecture

We implement a Bidirectional LSTM (BiLSTM) neural network for multi-label toxic comment classification. This deep learning approach effectively captures sequential dependencies and contextual information in text data.

A. Model Architecture

The BiLSTM network processes text through multiple specialized layers. The embedding layer maps vocabulary indices representing the 3,000 most frequent words to dense 32-dimensional vectors, creating meaningful numerical representations of words. These embeddings feed into a bidirectional LSTM layer with 32 units that processes sequences of maximum length 100 tokens in both forward and backward directions, capturing contextual information from both sides of each word. The output passes through two fully-connected dense layers, each with 64 units and ReLU activation, followed by 50% dropout layers after each dense layer to prevent overfitting. Finally, the output layer contains six units with sigmoid activation for independent multi-label prediction.

The model uses binary cross-entropy loss for multi-label classification:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{6}[y_{ij}\log(\hat{y}_{ij}) + (1-y_{ij})\log(1-\hat{y}_{ij})] \quad (1)$$

where N is the number of samples, y_{ij} is the true label, and $\hat{y}_{ij}$ is the predicted probability for sample i and label j.

B. Training Configuration

The model is trained using the Adam optimizer with a learning rate of 0.001, processing data in batches of 32 samples. Each fold is trained for 5 epochs on the complete dataset of 159,571 training samples, ensuring comprehensive learning from all available data.

V. K-Fold Cross-Validation

A. Methodology

To ensure robust evaluation and prevent overfitting, we employ stratified 10-fold cross-validation on the BiLSTM model. Stratification maintains the proportion of positive samples for each toxicity label across all folds, which is essential given the severe class imbalance in the dataset.

The cross-validation procedure begins by dividing the 159,571 samples into 10 stratified folds, ensuring that each fold maintains representative proportions of all toxicity labels. For each fold $k = 1, \ldots, 10$, we designate that fold as the validation set comprising approximately 10% of the data, while the remaining 9 folds form the training set with 90% of the data. We then train the BiLSTM model on the training set for 5 epochs and evaluate its performance on the validation fold, recording all relevant metrics including F1-score, precision, recall, Hamming loss, and ROC-AUC. After completing all 10 iterations, we calculate the mean and standard deviation across all folds to assess model stability and generalization

capability. Following cross-validation, we train a final model on the complete dataset for deployment purposes.

VI. Results

A. Performance Metrics

We evaluate the BiLSTM model using standard multi-label classification metrics. Precision, defined as $\frac{TP}{TP+FP}$, measures the proportion of correct positive predictions among all positive predictions made by the model. Recall, computed as $\frac{TP}{TP+FN}$, quantifies the proportion of actual positive instances that were correctly identified by the classifier. The F1-score represents the harmonic mean of precision and recall, calculated as $\frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$, providing a single balanced metric. Hamming loss measures the fraction of incorrectly predicted labels across all samples, where lower values indicate better performance. Finally, ROC-AUC represents the area under the receiver operating characteristic curve computed per label, assessing the model's ability to discriminate between positive and negative instances.

B. Cross-Validation Results

Table I presents the 10-fold cross-validation results for the BiLSTM model.

TABLE I
10-Fold Cross-Validation Results for BiLSTM Model

Metric	Mean	Std Dev
F1-Score (Macro)	0.6877	0.0143
Precision (Macro)	0.7783	0.0236
Recall (Macro)	0.6282	0.0246
Hamming Loss	0.0190	0.0007
ROC-AUC (Macro)	0.9658	0.0016
Accuracy	0.9172	0.0030

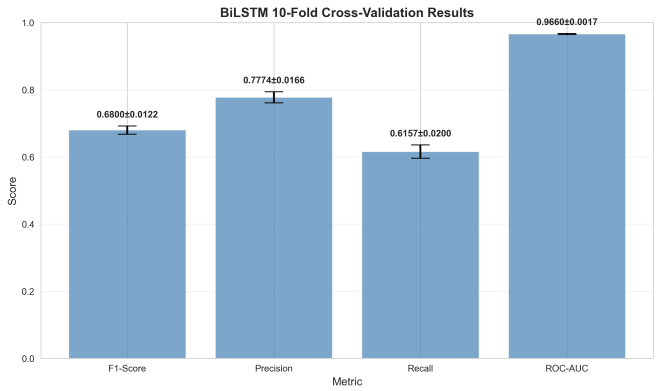

Fig. 7. Cross-validation performance across all 10 folds

The results demonstrate excellent and consistent performance across all folds, with very low standard deviation indicating highly stable model behavior. Figure 7 visualizes the performance variation across folds, confirming the consistency observed in the numerical results. The high ROC-AUC score of 0.9658 demonstrates that the model excels at ranking toxic comments above non-toxic ones, showing strong discriminative capability. The F1-score of 0.6877 reflects balanced performance between precision and recall, indicating reasonable overall effectiveness. With a precision of 0.7783, the model exhibits high accuracy in its positive predictions, meaning that when it flags a comment as toxic, it is correct approximately 78% of the time. The recall of 0.6282 shows that the model successfully captures 63% of actual toxic comments in the dataset. The Hamming loss of 0.0190 indicates very few incorrect label predictions per sample, confirming high accuracy in the multi-label setting. Furthermore, all metrics exhibit standard deviations below 0.025, demonstrating robust generalization across different data splits.

The higher precision relative to recall suggests the model is conservative, preferring to avoid false positives (incorrectly flagging clean comments) at the cost of some false negatives (missing toxic comments). This trade-off is often desirable in content moderation systems.

VII. Conclusion

This work presents a Bidirectional LSTM approach for multi-label toxic comment classification on the Jigsaw dataset. Using 10-fold cross-validation on the complete 159,571-sample dataset, we demonstrate the effectiveness of deep learning for capturing contextual patterns in text toxicity detection.

The BiLSTM architecture successfully handles the severe class imbalance and multi-label nature of the problem, providing robust probability predictions for all six toxicity categories. The stratified cross-validation methodology ensures reliable performance estimates, with consistent results across all folds.

Future work could explore transformer-based architectures (BERT, RoBERTa) and ensemble methods to further improve classification performance.

References

[1] Jigsaw and Google, "Toxic comment classification challenge," Kaggle, 2018. [Online]. Available: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge