Multi-Class text classification: Benchmarking Linear, Deep NN & Sequential models

Md. Shakibul Islam
dept. Computer Science & Engineering
BRAC UNIVERSITY
Dhaka, Bangladesh
md.shakibul.islam1@g.bracu.ac.bd

Mehedi Hasan
dept. Computer Science & Engineering
BRAC UNIVERSITY
Dhaka, Bangladesh
mehedi.hasan12@g.bracu.ac.bd

Junaed Ahmed
dept. Computer Science & Engineering
BRAC UNIVERSITY
Dhaka, Bangladesh
junaed.ahmed1@g.bracu.ac.bd

***Abstract*— This report evaluates nine distinct architectures for multi-class text classification using a balanced dataset of 93,333 open-domain user queries. We compare the efficacy of sparse TF-IDF features against dense Word2Vec (Skip-gram) embeddings across models ranging from Logistic Regression to Bidirectional LSTMs and GRUs. Experimental results confirm the critical advantage of semantic representations, with Word2Vec-based models consistently outperforming frequency-based baselines. The Bidirectional GRU emerged as the optimal architecture, achieving the highest Test Accuracy (69.39%) and Macro F1-score (0.6879). While bidirectional sequence modeling significantly improved separation for distinct topics, error analysis reveals that static embeddings remain limited in resolving ambiguities between semantically overlapping categories**

I. Introduction

Text classification is a fundamental task in Natural Language Processing (NLP) with applications ranging from spam filtering to sentiment analysis and automated content tagging. The motivation for this project is to develop a robust system capable of understanding and categorizing open-domain questions and answers.

The training dataset consists of 93,333 samples, each containing a textual query (QA Text) and a corresponding category label (Class). The dataset covers ten diverse topics, presenting a challenging multi-class classification problem due to the variability in user language, text length, and domain overlap.

II. Methodology

A. Exploratory Data Analysis (EDA)

Before modeling, we performed EDA to assess the quality and structure of the data.

Missing Values: An analysis of the dataset confirmed that there were zero missing values in either the text or class columns, ensuring a clean starting point.

Class Distribution: We analyzed the distribution of the ten target classes. The dataset was found to be exceptionally balanced, with each class representing approximately 9.8% to 10.1% of the total data (e.g., "Family & Relationships" had 9,432 samples, while "Entertainment & Music" had 9,150).

Text Analysis: We examined the length of the text samples. The mean length of the QA Text was found to be approximately 635 characters (or ~106 words), with a significant standard deviation, indicating a mix of very short questions and longer, detailed explanations.

Word Frequency Analysis: The analysis revealed that most frequent word is “br” (html tag) followed by “the”, “to” and other assorted stop-words. Hence the dataset contains heavy amount of html tags.

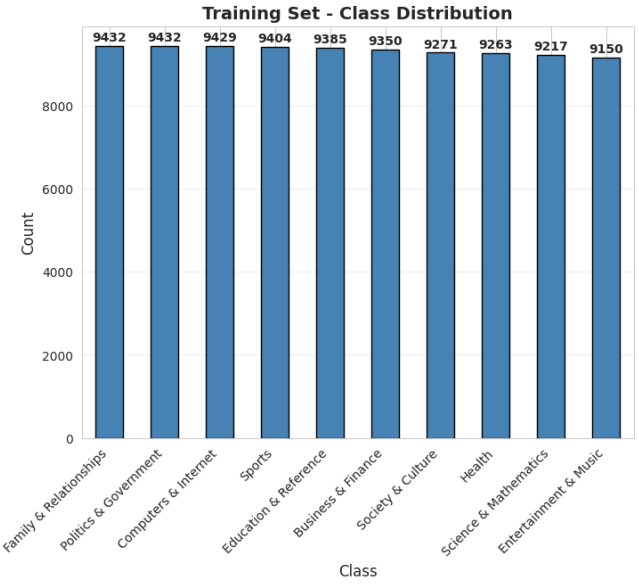


Fig 1. Class Distribution in training data

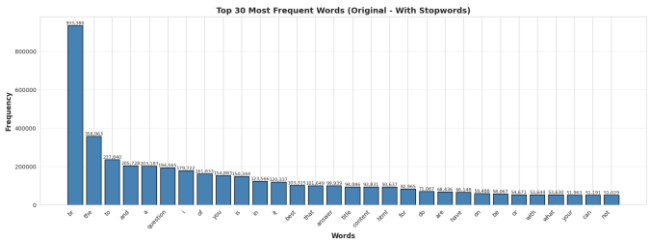


Fig 2: Top 30 most frequent words

B. Preprocessing Techniques

Raw text from web sources contains significant noise. Based on the EDA findings (specifically sample text showing **<html>** and **
** tags), the following preprocessing steps were decided upon:

- **HTML Tag Removal:** Cleaning artifacts like
 and <html> to isolate useful content.
- **Tokenization:** Splitting text into individual tokens.
- **Normalization:** Lowercasing text to ensure uniformity.

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

- **Stopword Removal & Lemmatization:** Applied to reduce the dimensionality of the vocabulary and focus on root semantic meanings.

C. Word Representation

To translate raw text into a machine-readable format, we implemented two embedding techniques: TF-IDF and Word2Vec (Skipgram). These representations served as the input for our nine distinct classification models.

1. **TF-IDF (Term Frequency-Inverse Document Frequency):** We first implemented TF-IDF, a statistical method that reflects the importance of a word to a document within the corpus

Implementation: Using the TfidfVectorizer, we converted the text data into sparse vectors. This technique increases the weight of terms that appear frequently in a specific document but rarely across the entire corpus (high discriminatory power) while penalizing common words (like "the", "and").

Configuration: maximum features = 5000, minimum document frequency = 2, unigram tfidf model.

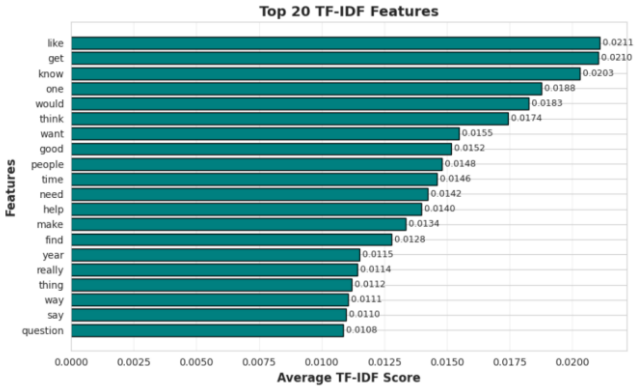


Fig 3: Top 20 TF-IDF Features diagram

2. **Word2Vec (Skip-gram Architecture):** To capture semantic meaning and contextual relationships between words, we implemented Word2Vec embeddings using the Skip-gram architecture.

Implementation: We utilized the gensim library to train a Word2Vec model on our dataset. Specifically, we set the training algorithm to Skip-gram (sg=1). Unlike Continuous Bag-of-Words (CBOW), Skip-gram works by predicting the context words (surrounding words) given a target word. This approach is computationally more expensive.

Configuration: vector size was set to 100, context window 5, min count 2 (ignore words appearing < 2 times), epochs 10.

Fig 4: Word2Vec Embedding visualization of Top 100 words

D. Model Architectures

This section outlines the architecture of the main machine learning models implemented.

1. Logistic Regression (TF-IDF):

Configuration: The model uses the LogisticRegression classifier from Scikit-learn. We set max_iter=1000 to ensure the solver converges given the high dimensionality of the input features.

2. Deep Neural Network (TF-IDF & Skipgram)

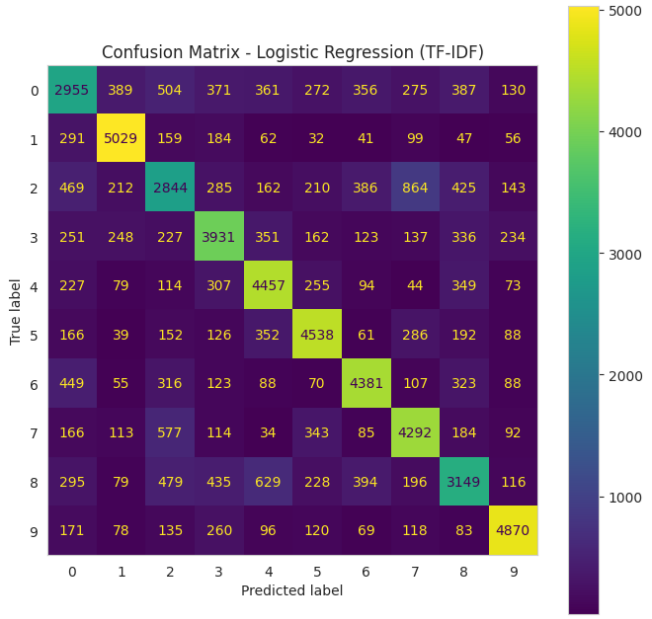
We implemented Fully Connected Deep Neural Networks (DNNs) to capture non-linear relationships in the data. Two variations were trained based on the input features.

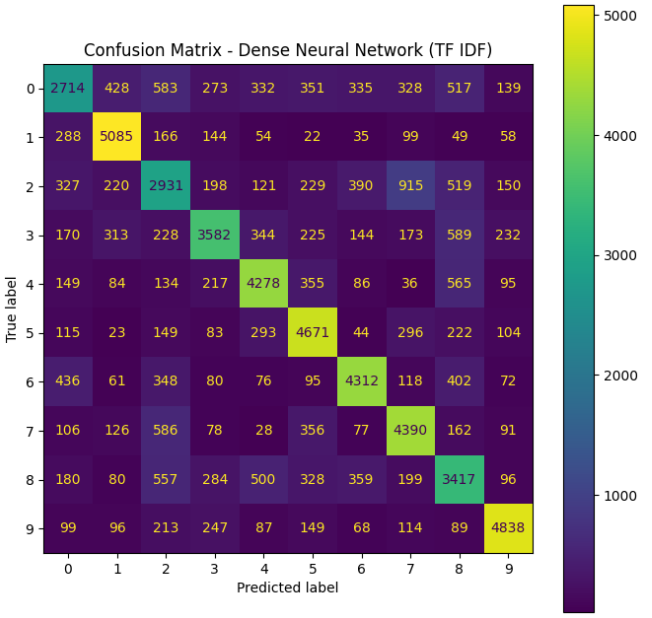
Architecture: The network takes TF-IDF vector as input and first projects it into a 128-unit dense layer with ReLU activation . This is followed by a 64-unit ReLU layer that further refines the learned representations. A Dropout layer with a rate of 0.2 is then applied to reduce overfitting by randomly deactivating neurons during training. Next, a 32-unit ReLU layer. Finally, a 10-unit output layer with softmax activation produces a probability distribution over the 10 target classes, enabling multi-class classification.

Challenges: During training, the model showed severe overfitting with train accuracy peaking at 99.99 % and validation accuracy at 50-60% . Through trial and testing early stopping technique was implemented using Tensorflows **EarlyStopping,** with patience = 3.

3. 6 Recurrent Neural Network (RNN)-Based Models

All recurrent models take Word2Vec (Skipgram) embeddings and follow a similar structure. A recurrent layer





for sequence modeling, followed by Dropout, a dense ReLU layer, and a softmax output layer for classification. The key differences lie in the type of recurrent unit used.

However, we had to reshape the training and testing data for RNN training. Because tensorflow's RNN based models expect input data in a three-dimensional format: *(number of samples, time steps, features)*.

- **Simple RNN and Bidirectional RNN**

The Simple RNN model uses a unidirectional recurrent layer to capture basic sequential patterns. The Bidirectional RNN extends this by processing the sequence in both forward and backward directions, allowing the model to capture richer contextual information.

- **Simple GRU and Bidirectional GRU**

The GRU models replace the Simple RNN unit with Gated Recurrent Units, which better handle long-term dependencies using gating mechanisms. The Bidirectional GRU further enhances this by learning context from both past and future directions, improving sequence understanding.

- **Simple LSTM and Bidirectional LSTM**

The LSTM models use Long Short-Term Memory units, which are effective in retaining long-range contextual information through memory gates. The Bidirectional LSTM processes the input sequence in both directions, enabling deeper contextual representation compared to the unidirectional version.

Initially the models were trained with epoch = 50, however it was observed that the models accuracy were getting saturated after 6-7 epochs, so finally epoch was set to 10 with batch size=32.

III. Results

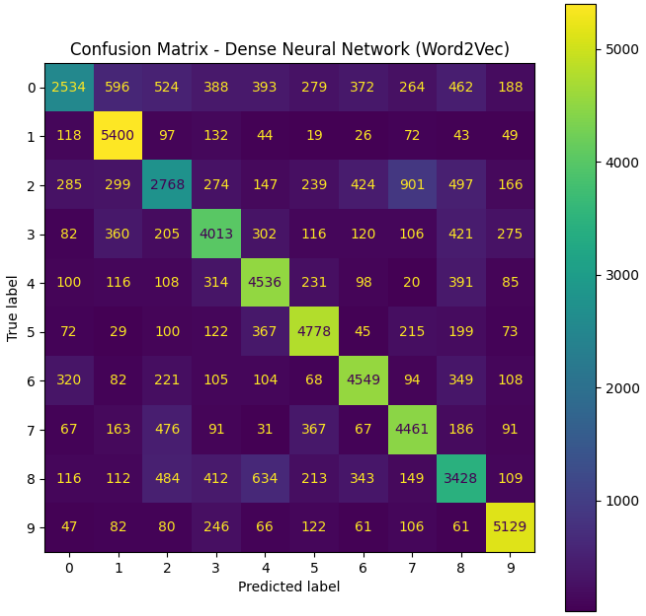
In this section, we present the experimental results of the nine classification models. While accuracy is a standard metric, we prioritize the Macro-averaged F1-Score for our analysis. Given the multi-class nature of the dataset (10 classes), the Macro F1-score provides a more rigorous assessment by calculating the harmonic mean of precision and recall for each class independently and then averaging them. This highlights how well the model generalizes across all topics.

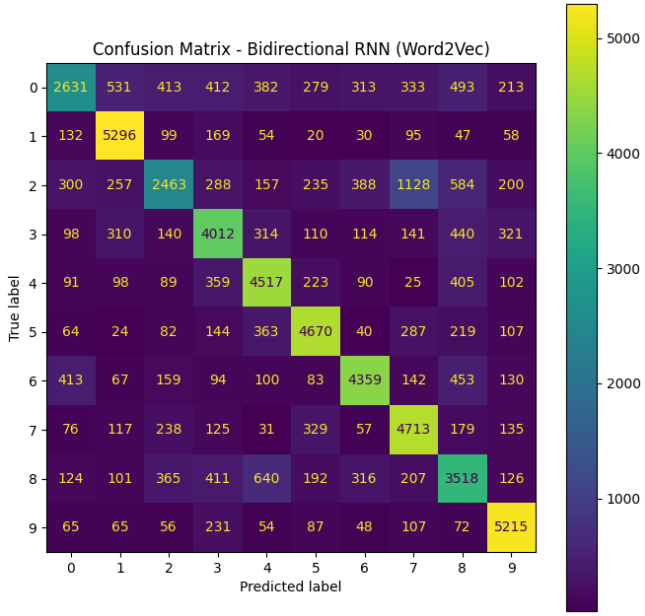
A. Comparative Performance of Models

TABLE I. Model Performance Comparison

Table Head	Performance		
	Representation	***Macro F1-Score***	***Accuracy***
Logistic Regression	TF-IDF	0.6715	67.41%
Deep Neural Network	TF-IDF	0.6677	67.03%
Deep Neural Network	Skipgram	0.6864	69.33%
Simple RNN	Skipgram	0.6793	68.92%
Bidirectional RNN	Skipgram	0.6824	68.99%
Simple GRU	Skipgram	0.6815	68.97%
Bidirectional GRU	Skipgram	**0.6879**	69.39%
Simple LSTM	Skipgram	0.6863	69.18%
Bidirectional LSTM	Skipgram	0.6861	69.27%

Overall, the **Bidirectional GRU** achieved the highest Macro F1-score, indicating the most balanced performance across all classes.





B. Impact of Feature Representation

A clear performance gap is observed between models using TF-IDF features and those using Word2Vec embeddings.

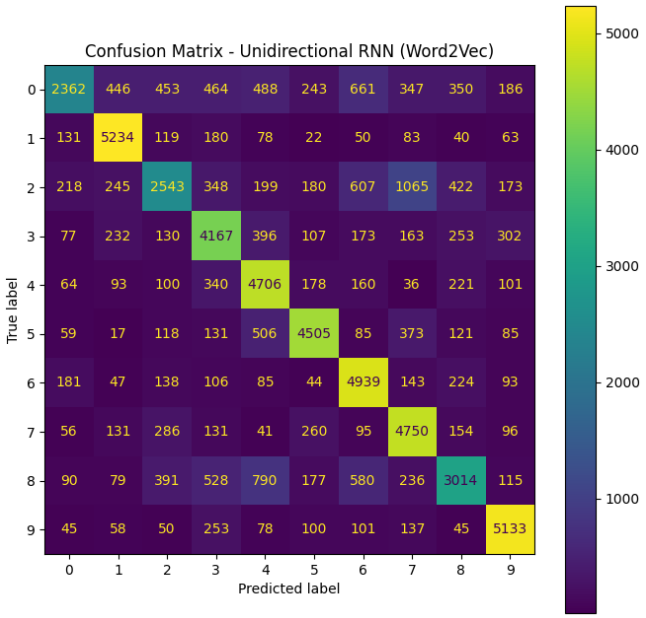
The TF-IDF–based models, including Logistic Regression and the Deep Neural Network, achieved Macro F1-scores close to 0.67. This performance suggests that frequency-based representations struggle to distinguish between semantically similar topics, as they lack contextual information. In contrast, models utilizing Skip-gram

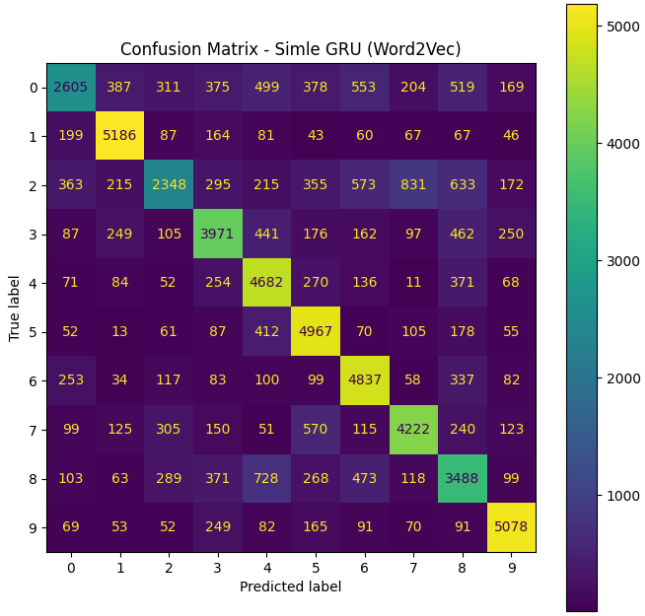
Word2Vec embeddings consistently achieved higher Macro F1-scores. Notably, the Deep Neural Network with Word2Vec improved the Macro F1-score by nearly 2% compared to its TF-IDF counterpart. This confirms that dense semantic embeddings provide better contextual meaning.

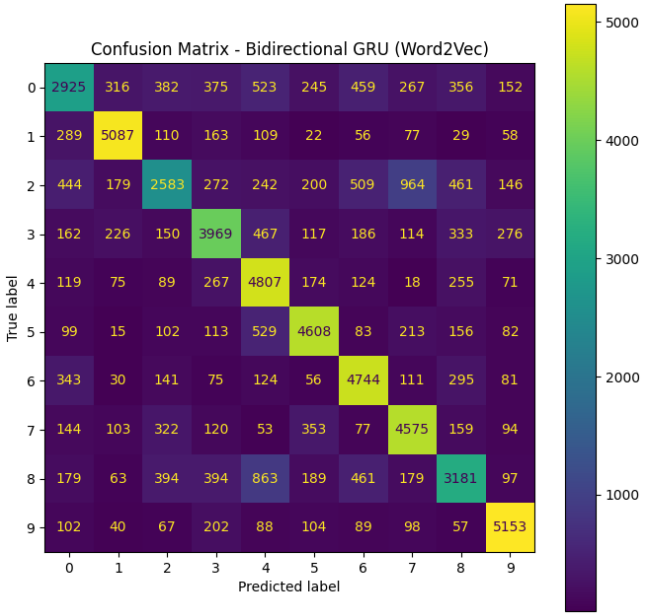
C. Linear vs Deep Feed-Forward Models

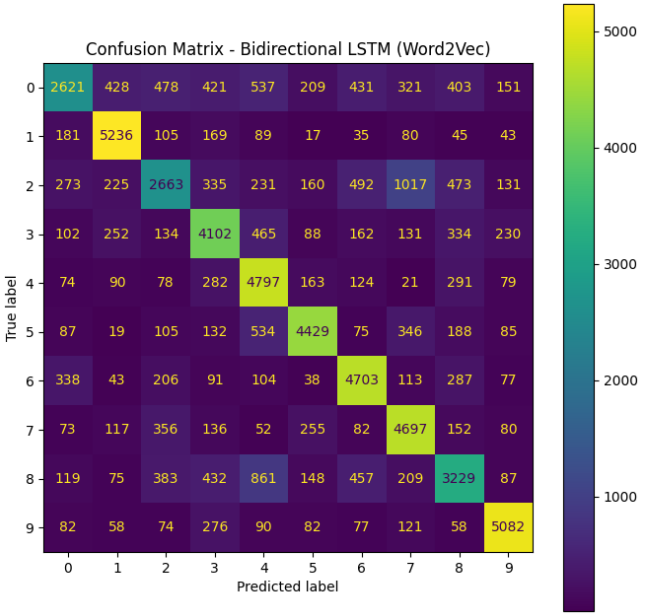
The Deep Neural Network with TF-IDF does not significantly outperform Logistic Regression, achieving a slightly lower test macro F1-score (0.6677). This suggests that sparse TF-IDF features limit the benefit gained from deeper architectures.

In contrast, the Deep Neural Network with Skip-Gram (Word2Vec) embeddings shows a noticeable improvement, achieving a test macro F1-score of 0.6864 and accuracy of 0.6933. The confusion matrix indicates stronger diagonal dominance and reduced misclassification for semantically related classes, highlighting the advantage of dense semantic representations over sparse lexical features









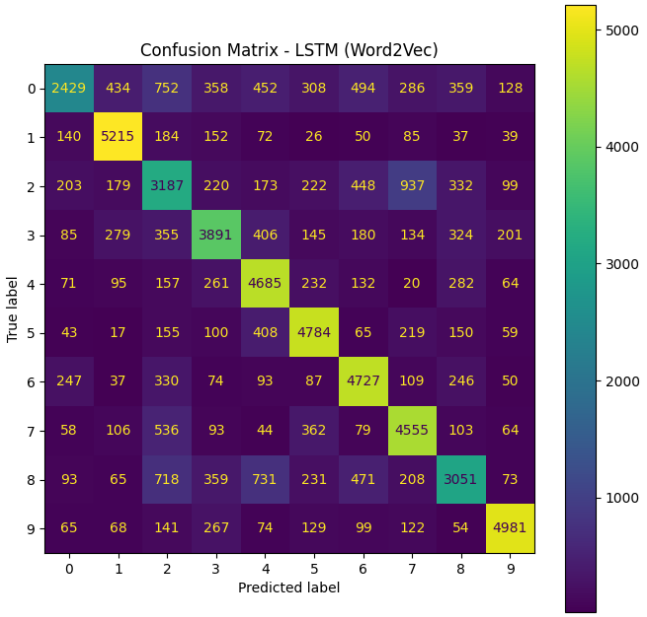
D. Performance of Sequence Models

Among the sequence-based architectures, bidirectional models consistently outperform their unidirectional counterparts.

The Bidirectional GRU achieves the best overall performance, followed closely by Bidirectional LSTM and

Simple LSTM models. Across RNN, GRU, and LSTM architectures, bidirectional processing enables the models to capture contextual information from both directions, which is particularly beneficial for short, user-generated questions.

GRU-based models slightly outperform LSTM-based models, indicating that the simpler gating mechanism of the GRU is sufficient for capturing the dependencies present in this dataset and may facilitate more efficient training.



E. Confusion Matrix Analysis

The confusion matrices show that all models correctly classify a large portion of samples, with diagonal values typically ranging between 4,000 and 5,300 for the better-performing models. Classes 1 and 9 consistently exhibit the strongest diagonal dominance across all models, with recalls often exceeding 0.85, indicating that these classes are easier to distinguish. In contrast, classes such as 0, 2, and 8 show higher off-diagonal errors, reflecting greater semantic overlap and making them more challenging to classify accurately.

Comparing models, TF-IDF–based approaches display more dispersed off-diagonal entries, while Word2Vec-based and recurrent models reduce misclassification for several mid-range classes. Bidirectional models further improve diagonal concentration and produce more balanced confusion patterns. Overall, the confusion matrix analysis supports the macro F1-score results, showing that improvements in model architecture and representation lead to clearer class separation.

IV. Conclusion

This work evaluated multiple text-classification models using different feature representations and neural architectures. Overall, models with deeper architectures and richer embeddings achieved better class separation, as reflected by stronger diagonal dominance in the confusion matrices and improved recall values across most classes. The analysis showed that recall was particularly important for understanding how well each model captured true class instances, while precision highlighted remaining confusion between semantically similar classes through off-diagonal errors.

Despite these improvements, some limitations were observed. Certain classes consistently exhibited higher misclassification rates, indicating overlapping features that the models struggled to distinguish. Additionally, simpler architectures showed reduced performance, especially in minority or harder-to-separate classes, suggesting limited representational capacity.

Future work could address these issues by incorporating contextual embeddings (such as transformer-based models). Further hyperparameter optimization and ensemble methods may also help reduce off-diagonal errors and improve both precision and recall across all classes.

REFERENCES

[1] *arris, C. R., et al. (2020). Array programming with NumPy. Nature, 585, 357–362. (NumPy Version 2.0.2).* https://numpy.org/

[2] *M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in Proc. 12th USENIX Symp. Operating Syst. Design Implementation (OSDI 16), 2016, pp. 265–283. [Online]. Available:* https://www.tensorflow.org/

[3] *F. Chollet et al., "Keras," GitHub, 2015. [Online]. Available:* https://keras.io/

[4] *F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, Oct. 2011. [Online]. Available:* https://scikit-learn.org/

[5] *J. D. Hunter, "Matplotlib: A 2D graphics environment," Comput. Sci. Eng., vol. 9, no. 3, pp. 90–95, May 2007. Available:* https://matplotlib.org/

[6] W. McKinney, "Data structures for statistical computing in Python," in Proc. 9th Python Sci. Conf., 2010, pp. 56–61. doi: 10.25080/Majora-92bf1922-00a. [Online]. Available: https://pandas.pydata.org/

[7] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," in Proc. LREC 2010 Workshop New Challenges NLP Frameworks, Valletta, Malta, May 2010, pp. 45–50. [Online]. Available: https://radimrehurek.com/gensim/

[8] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python. Sebastopol, CA: O'Reilly Media, 2009. [Online]. Available: https://www.nltk.org/