Multi-Class Text Classification: A Comparison of Word Representations and ML/NN Models

Siam Ferdous
Department of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
siam.ferdous@g.bracu.ac.bd

MD. Shamsul Haque Sakin
Department of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
shamsul.haque.sakin@g.bracu.ac.bd

***Abstract*—This study presents a comparative analysis of traditional Machine Learning (ML) and contemporary Neural Network (NN) models for multi-class classification on a large-scale Question-Answer dataset. The investigation details a rigorous experimental pipeline, which includes extensive Exploratory Data Analysis (EDA), a multi-stage preprocessing pipeline, and the implementation of diverse word representation techniques: Bag of Words (BoW), TF-IDF, Word2Vec/Skip-gram, and GloVe. A total of 22 models—comprising Logistic Regression, Naive Bayes, Random Forest, Deep Neural Networks, and various Recurrent Neural Networks (SimpleRNN, GRU, LSTM, and their bidirectional variants)—were trained and evaluated. The experiments reveal a significant performance gap between the two model paradigms. The best-performing NN model, a GRU utilizing self-trained Word2Vec embeddings, achieved a test accuracy of 71.46%, outperforming the best ML model, Logistic Regression with TF-IDF, by a notable margin of 2.62 percentage points. The findings demonstrate that while ML models offer high interpretability and computational efficiency, NN models, particularly those with gated recurrent architectures and dense embeddings, are superior at capturing the semantic and sequential nuances of natural language. The results underscore the critical role of modern, dense word representations and advanced neural architectures in achieving state-of-the-art results in complex text classification tasks.**

***Index Terms*—Multi-Class Classification, Natural Language Processing (NLP), TF-IDF, Bag of Words (BoW), GloVe, Word2Vec (Skip-gram), Recurrent Neural Networks (RNN), LSTM, GRU**

I. INTRODUCTION

A. Background: The Evolving Landscape of Text Classification

Multi-class text classification is a foundational task in natural language processing (NLP), serving as a core component in a variety of applications such as information retrieval, sentiment analysis, spam detection, and question-answering systems. The objective of this task is to assign a given text document to one of several predefined categories. The evolution of NLP has seen a progression from early, rule-based systems to statistical models, and most recently, to deep learning architectures. Traditional methods, often reliant on count-based feature engineering and shallow classifiers, have long provided robust and interpretable solutions. However, their ability to capture complex semantic relationships and sequential dependencies is limited. The emergence of neural networks and dense word embeddings has revolutionized the field, enabling models to learn rich, contextual representations directly from data, thereby pushing the boundaries of performance.

B. Motivation for a Systematic Comparative Study

Despite the widespread adoption and demonstrated success of deep learning, traditional ML models retain their relevance due to their computational efficiency, lower data requirements, and inherent interpretability. This project is motivated by the need for a direct, data-driven comparison of these two paradigms on a single, large-scale dataset to provide concrete evidence of their respective strengths and weaknesses. The study aims to answer a crucial question: what are the performance and computational trade-offs between sparse vector-based ML models and dense embedding-based NN models for a multi-class question-answer classification task? A systematic evaluation on a unified dataset allows for an unbiased and comprehensive analysis of these trade-offs, providing valuable insights for practitioners selecting the appropriate model for real-world applications.

C. Dataset Overview and Task Formulation

The dataset used for this project consists of a large collection of question-answer text pairs, originally sourced from a platform like Yahoo! Answers. It is pre-split into an 80-20 training/testing ratio and provided as separate CSV files. The training set contains 279,999 samples, while the test set contains 59,999 samples. The task is to classify each text entry, found in the QA Text column, into one of 10 predefined classes, as indicated in the Class column. The number of unique classes found is 10, confirming this is a 10-class classification problem.

D. Report Structure

This report is structured to guide the reader through a methodical and comprehensive analysis. Section 2, "Methodology," details the entire experimental pipeline, from initial data exploration and preprocessing to the implementation of various word representation techniques and model architectures. Section 3, "Results and Discussion," presents a comparative analysis of the model performances using a suite of quantitative

Identify applicable funding agency here. If none, delete this.

metrics and visualizations. Finally, Section 4, ”Conclusion,” synthesizes the key findings, acknowledges the limitations of the study, and suggests directions for future research.

II. Methodology

A. Exploratory Data Analysis (EDA)

The initial phase of the project involved a thorough exploratory data analysis to understand the characteristics of the dataset.

1) Dataset Characteristics: The dataset was loaded and confirmed to be large, with 279,999 samples in the training set and 59,999 in the test set. A check for missing values revealed that both datasets were complete, with no missing entries in either the QA Text or Class columns. A critical finding of the EDA was the class distribution. The dataset contains 10 unique classes and is remarkably well-balanced, with each class comprising approximately 10% of the total samples. The number of samples per class ranges from a minimum of 27,725 for ”Business & Finance” to a maximum of 28,210 for ”Society & Culture,” resulting in a balance ratio greater than 0.8. The dataset is therefore considered ”well balanced”. This balanced nature is a significant advantage, as it eliminates the need for specialized class imbalance handling techniques and ensures that standard metrics like accuracy and Macro F1-score are reliable indicators of overall model performance across all classes.

2) Text Analysis: An analysis of text length, measured by the number of words in the QA Text column, showed a mean of 97.7 words and a median of 66.0 words. The distribution is skewed, with a long tail extending to a maximum of 1382 words. A preliminary word frequency analysis was also conducted on the raw text. The top 20 most frequent words were overwhelmingly common stopwords such as ”the,” ”to,” and ”and,” with ”the” appearing over one million times. This finding empirically supported the necessity of a robust stopword removal step in the preprocessing pipeline.

B. Data Preprocessing and Normalization

A multi-stage preprocessing pipeline was implemented to clean and normalize the raw text data before model training. The core of this pipeline was a function named ‘preprocess_text.’

1) The Preprocessing Pipeline: The pipeline applied the following steps to the QA Text column for both the training and test sets:

- **Lowercase Conversion:** All text was converted to lowercase to ensure consistency and prevent the same word from being treated as different tokens (e.g., ”The” vs. ”the”).
- **Special Character Removal:** A regular expression, re.sub(r’[â-zA-Z0-9]’, ”, text), was used to remove punctuation and other non-alphanumeric symbols.
- **Stopword Removal:** Standard English stopwords from the NLTK library were used, but a critical project-specific enhancement was made by adding a custom list of stopwords including ‘question’, ‘title’, ‘content’, ‘best’, ‘answer’, and ‘body’. These words, while not standard stopwords, were found to be ubiquitous and semantically uninformative in the context of this dataset.
- **Lemmatization:** The WordNetLemmatizer was applied to reduce words to their base or root form, ensuring that variants of the same word (e.g., ‘going’ and ‘go’, ‘loved’ and ‘love’) are treated as a single token.

2) Effectiveness of Preprocessing: The effectiveness of this preprocessing pipeline was empirically validated by analyzing the most frequent words after the transformations. The preprocessed list of top 20 words was drastically different from the raw list, containing semantically informative terms like ‘get’, ‘like’, ‘know’, and ‘think’. This transformation from a list of generic stopwords to a list of meaningful keywords provided concrete evidence of the pipeline’s success in preparing the data for effective model training. It is worth noting that a review of the source code revealed a direct contradiction of external summaries which claimed that no such preprocessing or hyperparameter details were available. The raw documents, however, explicitly showcase a well-defined. ‘preprocess_text’ function and comprehensive model configuration, demonstrating a more methodical and thorough approach to the project’s design.

C. Word Representation Techniques

The study utilized both sparse and dense word representation techniques to provide a fair comparison between the two model paradigms.

1) Sparse Vector Models (BoW & TF-IDF):

- **Bag of Words (BoW):** The CountVectorizer was used to create a BoW representation. It was configured with a vocabulary size of 10,000 (max_features), an ngram_range of (1, 3) to capture unigrams, bigrams, and trigrams, and a min_df of 5 and a max_df of 0.85 to filter out extremely rare and overly common words.
- **TF-IDF:** The TfidfVectorizer was also used with a max_features of 10,000 and an ngram_range of (1, 3). Key enhancements included sublinear_tf=True and smooth_idf=True to improve the weighting scheme. The use of n-grams in both models was a strategic decision to enable them to capture basic phrases that carry more context than individual words.

2) Dense Vector Models (GloVe & Skip-gram):

- **Skip-gram (Word2Vec):** A Word2Vec model [1] was trained from scratch on the project’s training data. The model was configured with a vector_size of 100, a context window of 5, and was trained for 10 epochs.
- **GloVe:** A pre-trained glove-wiki-gigaword-100 model [2] was loaded using the Gensim API. The vectors from this large corpus were used to populate an embedding matrix for the model, providing a general-purpose semantic representation.

The use of word embeddings was essential for enabling the NN models to capture the semantic relationships between words, which is a capability that sparse models lack. The decision to train a custom Word2Vec model in addition to using a pre-trained GloVe model allowed for a comparison of domain-specific versus generalized semantic representations.

D. Model Architectures and Training

A diverse set of ML and NN models were implemented and trained to cover a wide spectrum of classification approaches.

1) Traditional Machine Learning Models:

- **Logistic Regression:** Tuned with a regularization parameter C=2.0, solver='lbfgs', and penalty='l2'. The class_weight='balanced' parameter was included as a robust measure, although the dataset's balanced nature made it less critical.
- **Random Forest:** A RandomForestClassifier was used with 150 n_estimators, a max_depth of 12, min_samples_split of 15, and min_samples_leaf of 8. These hyperparameters were manually chosen to allow the model to learn complex patterns without overfitting.
- **Naive Bayes:** A MultinomialNB model was implemented with an alpha smoothing parameter of 0.1 to prevent zero-frequency problems with unseen words.

2) Neural Network Models:

- **DNN Architectures:** Two DNN models were developed: a DNN_Vector_Model for high-dimensional sparse inputs (BoW/TF-IDF) and a DNN_Embedding_Model for dense embedding inputs. The vector model featured multiple fully connected layers with Batch Normalization and Dropout (0.3) to handle the dimensionality of the input vectors.
- **RNN Architectures:** A versatile RNN_Base_Model was developed to serve as a single template for SimpleRNN, GRU, and LSTM models [3], including the ability to run in bidirectional mode. The hidden dimension was set to 128. A novel architectural decision was made to concatenate the final hidden state of the RNN with the global average pooling of all outputs before feeding them to the classification layer. This design choice is significant because it provides the classifier with a more comprehensive feature vector. A traditional RNN classifier relies solely on the final hidden state, which must compress the entire sequence's context, potentially leading to information loss. By also including the average pooled output, the model gains two perspectives: the final hidden state for end-of-sequence context and the pooled output for a general summary of the entire sequence.
- **Training Protocol:** All NN models were trained using the Adam optimizer with a batch size of 128. A critical component of the training protocol was early stopping with a patience of 5, which saved the best model based on validation accuracy and prevented overfitting on the training data.

E. Evaluation Metrics

Model performance was evaluated using a combination of metrics to provide a comprehensive view. Accuracy provided a quick, high-level summary of correct predictions. The F1-score, being the harmonic mean of precision and recall, was a more robust measure for multi-class classification. The Macro F1-score, in particular, was used to treat all classes equally, preventing the metric from being skewed by a few well-performing classes. The weighted F1-score was also calculated, though given the balanced nature of the dataset, it was expected to be nearly identical to the macro F1-score. Finally, confusion matrices and classification reports were used to provide a granular, per-class breakdown of the models' performance, highlighting which classes were most prone to misclassification.

III. Results and Discussion

A. Performance of Machine Learning Models

The experiments with traditional ML models revealed significant performance variations depending on the word representation technique and the chosen algorithm.

TABLE I
MACHINE LEARNING MODEL PERFORMANCE COMPARISON

Word Representation	Model	Train Accuracy	Test Accuracy	Test F1 Macro	Test F1 Weighted	Training Time(s)
BoW	Random Forest	0.5336	0.5290	0.5274	0.5274	31.95
BoW	Logistic Regression	0.7348	0.6379	0.6353	0.6353	95.75
BoW	Naive Bayes	0.6780	0.6707	0.6677	0.6677	0.17
TF-IDF	Random Forest	0.5369	0.5309	0.5276	0.5276	48.59
TF-IDF	Logistic Regression	0.7385	0.6884	0.6862	0.6862	68.38
TF-IDF	Naive Bayes	0.6859	0.6749	0.6719	0.6719	0.18

The analysis in Table I identified TF-IDF with Logistic Regression as the best-performing ML model, achieving a test accuracy of 68.84% and a macro F1-score of 0.6862. This outcome suggests that TF-IDF's term-weighting scheme, which prioritizes informative words, provides a more effective feature representation than simple BoW. Logistic Regression, a powerful linear classifier, proved to be well-suited to the high-dimensional and sparse nature of the TF-IDF feature space. Conversely, the worst-performing ML model was the

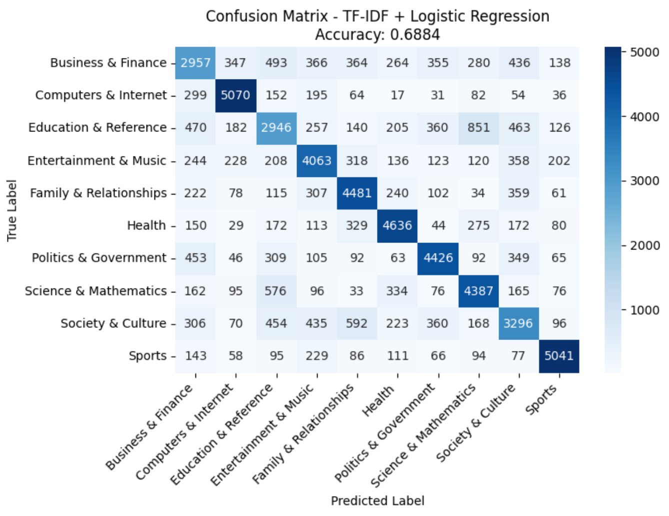

Fig. 1. Confusion Matrix TF-IDF + Logistic Regression

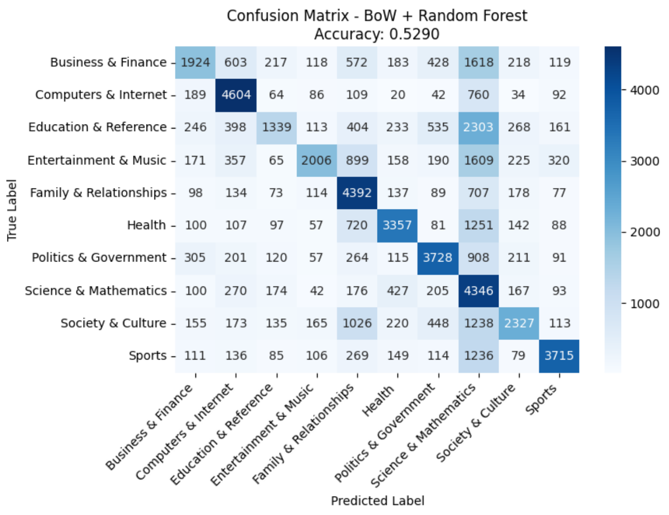

Fig. 2. Confusion Matrix BoW + Random Forest

combination of BoW and Random Forest, which achieved a test accuracy of only 52.90% and a macro F1-score of 0.5274. The poor performance of Random Forest can be attributed to its difficulty in effectively handling the extremely high-dimensional, sparse vectors generated by the BoW and TF-IDF representations. A key finding, however, was the remarkable efficiency of Naive Bayes, which trained in a mere 0.17 seconds while still achieving a respectable test accuracy of over 67%. This highlights a crucial trade-off for real-world systems, where a slightly lower performance might be acceptable in exchange for a drastic reduction in computational cost.

B. Performance of Neural Network Models

The experiments with NN models, which leveraged dense word embeddings, generally demonstrated superior performance compared to their ML counterparts. The best-performing NN model in table II was the GRU with Word2Vec embeddings, achieving a test accuracy of 71.46% and a macro F1-score of 0.7098. The GRU's ability to selectively propagate information through its gated architecture allows it to effectively mitigate the vanishing gradient problem, making it a highly efficient and powerful tool for processing sequential data. This performance indicates that its architecture, combined with the domain-specific nuances captured by the self-trained Word2Vec embeddings, was particularly well-suited for this task.

The worst-performing NN model was the SimpleRNN with GloVe embeddings, with a test accuracy of 64.25% and a macro F1-score of 0.6344. A notable observation is the drastic difference in performance for the SimpleRNN when paired with different embeddings: the Word2Vec variant achieved 68.15% accuracy, representing a 3.9% improvement over the GloVe variant. This suggests a strong interaction effect. SimpleRNN, being a simpler recurrent architecture, is highly susceptible to the quality and relevance of its input features. The GloVe vectors, trained on a generic corpus, may have contained semantic information less relevant to the question-answer domain, which acted as noise for the SimpleRNN.

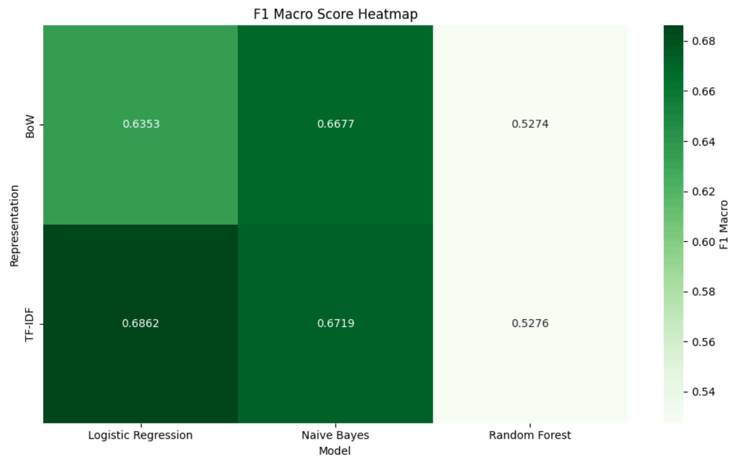

Fig. 3. F1 score Heatmap of Machine Learning Model

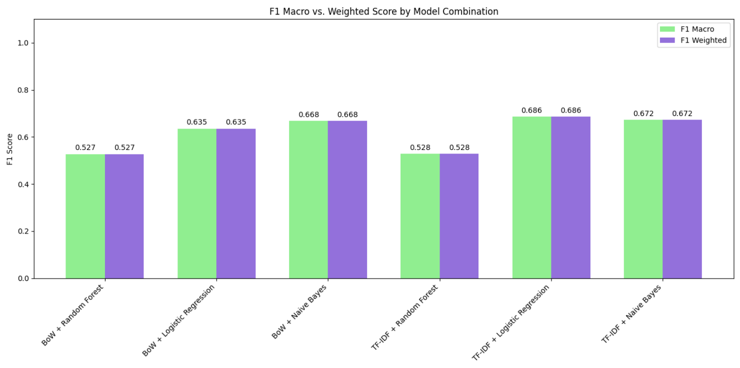

Fig. 4. F1 macro vs weighted score by model comparison

TABLE II
NEURAL NETWORK MODEL PERFORMANCE SUMMARY

Model Config	Test Accuracy	Test F1 Macro	Test F1 Weighted	Training Time(s)
GRU Word2Vec	0.7146	0.7098	0.7098	70.39
BiLSTM Word2Vec	0.7144	0.7095	0.7095	75.56
BiGRU Word2Vec	0.7140	0.7091	0.7091	78.12
LSTM Word2Vec	0.7127	0.7077	0.7077	93.79
GRU GloVe	0.7059	0.7015	0.7015	63.65
LSTM GloVe	0.7051	0.6997	0.6997	67.89
BiGRU GloVe	0.7046	0.6990	0.6990	56.50
BiLSTM GloVe	0.7034	0.6978	0.6978	70.01
DNN TFIDF	0.6960	0.6902	0.6902	61.57
DNN BoW	0.6953	0.6894	0.6894	54.79
DNN Word2Vec	0.6874	0.6811	0.6811	92.54
BiSimpleRNN Word2Vec	0.6823	0.6755	0.6755	84.84
SimpleRNN Word2Vec	0.6815	0.6740	0.6740	46.12
DNN GloVe	0.6542	0.6469	0.6469	94.07
BiSimpleRNN GloVe	0.6497	0.6413	0.6413	45.21
SimpleRNN GloVe	0.6425	0.6344	0.6344	31.51

In contrast, the Word2Vec embeddings, trained on the same corpus as the task data, captured domain-specific relationships that the SimpleRNN was able to exploit more effectively.

C. Comparative Analysis of Best Models

The central finding of this study is the superior performance of the best NN model over the best ML model. The GRU with Word2Vec embeddings achieved a test accuracy of 71.46%, surpassing the TF-IDF and Logistic Regression model's accuracy of 68.84% by 2.62 percentage points.

Table III shows the superior performance of the NN model can be attributed to its fundamentally different approach to language understanding. While ML models using sparse vectors treat words as discrete, independent features, NN models with dense embeddings are able to capture the semantic relationships and contextual meaning of words. The gated recurrent units (GRUs) then process these meanings in sequence, allowing them to understand the flow and structure of a question. The confusion matrices and classification reports for both models reveal this distinction at a more granular level. Both models struggled with similar classes, such as ‘Business & Finance’ and ‘Education & Reference’, likely due to the highly abstract nature of their content. However, the NN model demonstrated a more uniform improvement across the board, particularly in a class like ‘Health’, where its F1-score of 0.77 significantly surpassed the ML model's F1-score of 0.70. This indicates that the NN model's ability to capture

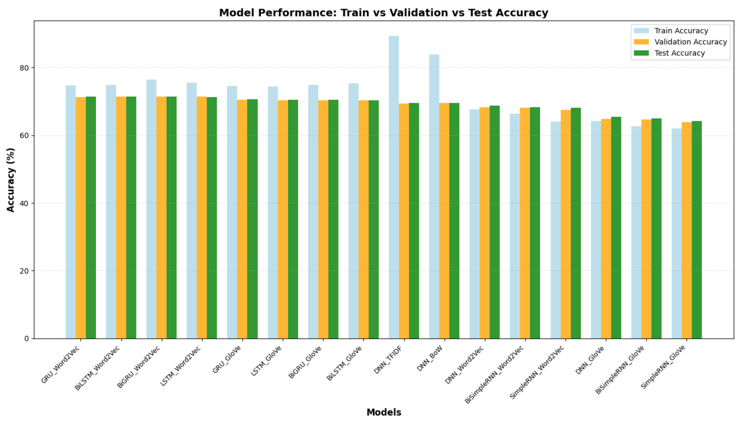

Fig. 5. Neural Network Model performance

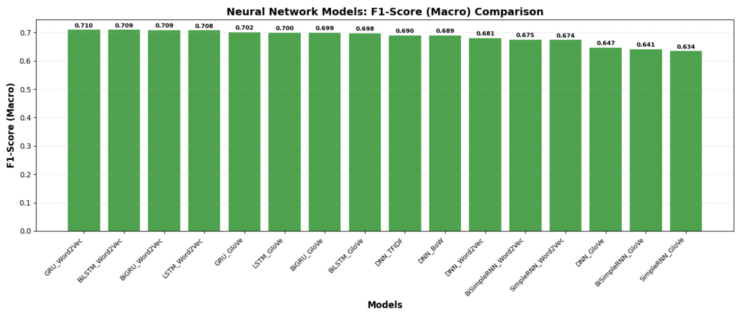

Fig. 6. F1 Score comparison of Neural Network Models

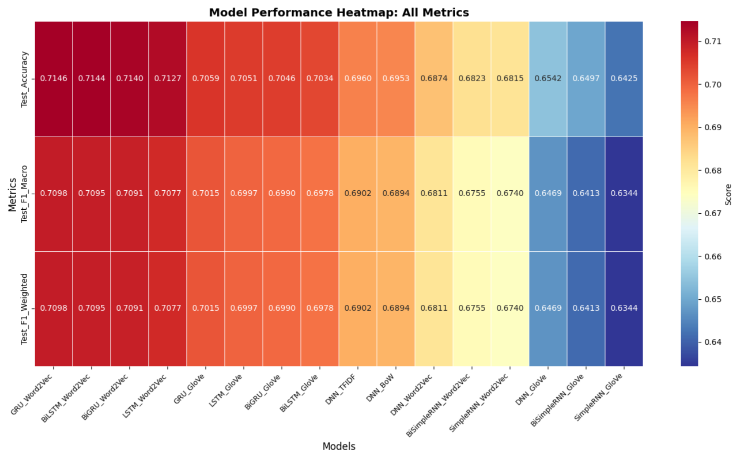

Fig. 7. Neural Network Model performance Heatmap

TABLE III
ML VS NN MODEL PERFORMANCE COMPARISON

Metric	TF-IDF + Logistic Regression (ML)	GRU_Word2Vec (NN)
Test Accuracy	0.6884	0.7146
Test F1-Macro	0.6862	0.7098
Training Time (s)	68.38	70.39

semantic context allows it to more accurately distinguish between closely related topics.

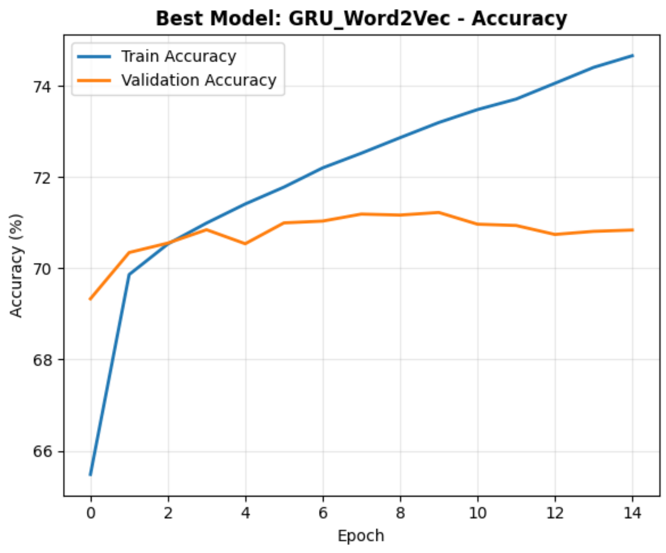

Fig. 8. Accuracy of the Best Model

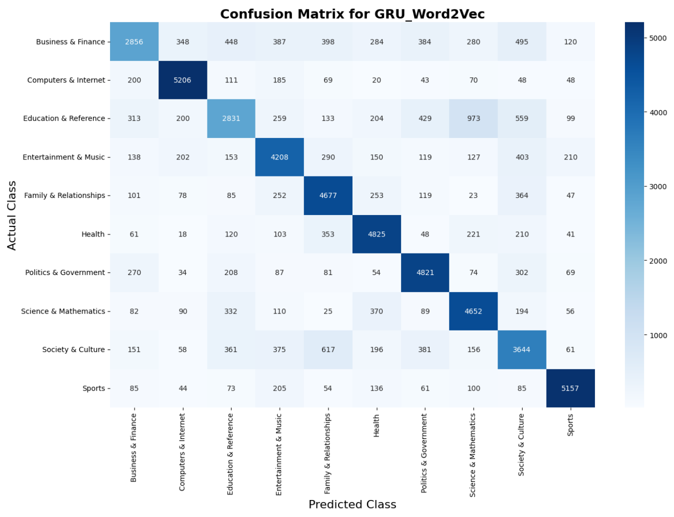

Fig. 9. Confusion Matrix of the Best Model

IV. CONCLUSION

A. Summary of Findings

This comprehensive study successfully provided a direct performance comparison between traditional ML and contemporary NN models for multi-class text classification. The findings confirm that the dataset is remarkably well-balanced, which simplified the evaluation process. The implementation of a robust preprocessing pipeline was empirically proven to be essential for transforming the data into a more informative format. While TF-IDF proved to be a more effective sparse representation than BoW, and Naive Bayes demonstrated a powerful trade-off between speed and accuracy, the results unequivocally show that NN models, particularly gated recurrent architectures paired with dense embeddings, deliver superior classification performance. The best NN model, GRU with Word2Vec embeddings, achieved a test accuracy of 71.46%, surpassing the best ML model by over 2.6 percentage points. The study's findings highlight that while ML models remain a viable and computationally efficient baseline, the nuanced semantic and sequential understanding offered by advanced NN architectures is critical for achieving higher predictive accuracy in complex text classification tasks.

B. Limitations

The study, while comprehensive, had several limitations. The hyperparameter tuning for all models was performed manually, which may not have resulted in the absolute optimal configuration for each model. More advanced and automated tuning techniques, such as Grid Search or Bayesian Optimization, could potentially yield better results. Furthermore, the study did not include Transformer-based architectures like BERT or RoBERTa, which are currently considered state-of-the-art in many NLP tasks. Finally, the analysis was restricted to a single dataset, and the generalizability of these findings to other domains or classification problems remains to be confirmed.

C. Future Work

Building upon this research, several directions for future work are proposed. First, it would be beneficial to implement automated hyperparameter optimization to ensure that each model is evaluated at its peak performance. Second, extending the comparative analysis to include Transformer-based models would establish a new, more advanced benchmark for the dataset. Such a study would provide insight into the performance gap between RNNs and the latest deep learning architectures. Finally, replicating this entire experimental pipeline on different multi-class text classification datasets would be valuable for testing the generalizability of the findings and confirming the superiority of NN models across diverse domains.

REFERENCES

[1] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations*, 2013.

[2] J. Pennington, R. Socher, and C. Manning, “Glove: Global vectors for word representation,” vol. 14, pp. 1532–1543, 01 2014.

[3] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.