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

Project
Natural Language Processing

Name:

Md. Adnan Parvez
Nowshin Reza

***Abstract*—This project explores multi-class text classification using different word representation techniques and both machine learning (ML) and neural network (NN) models. The dataset, already split into 80% training and 20% testing, was thoroughly analyzed and preprocessed. This included lowercasing, tokenizations, and removing common stopwords for both training and testing data.Four word representation approaches were implemented: Bag of Words (BoW), TF-IDF, GloVe, and Skip-gram. A total of 22 experiments were conducted across ML models (Random Forest, Logistic Regression, Naive Bayes) and NN architectures (Deep Neural Network, SimpleRNN, GRU, LSTM, Bidirectional SimpleRNN, Bidirectional GRU, and Bidirectional LSTM). The results show that GloVe + LSTM, a neural network model, performed the best (Accuracy: 0.7138, F1-Macro: 0.7081, F1-Weighted: 0.7081), while TF-IDF + Random Forest, a traditional ML model, performed the worst (Accuracy: 0.4893, F1-Macro: 0.4968, F1-Weighted: 0.4968). This shows that context-aware embeddings and sequential neural networks work much better than simple frequency-based methods for multi-class text classification.**

***Index Terms*—Multi-class text classification, word embeddings, machine learning, neural networks, preprocessing, tokenizations, BoW, TF-IDF, GloVe, Skip-gram, Random Forest, LSTM, accuracy, F1-score, context-aware embeddings.**

I. Introduction

Text classification is an important task in Natural Language Processing (NLP) used in spam detection, sentiment analysis, and topic categorization. Traditional methods like Bag of Words and TF-IDF focus on word frequency, while embeddings such as GloVe and Skip-gram capture meaning and context. This project evaluates how different word representations work with various ML and NN models in a multi-class setting. We conducted 22 experiments to compare models and representations, highlighting differences in accuracy, efficiency, and robustness.

II. Methodology

The project followed a structured pipeline comprising exploratory data analysis (EDA), preprocessing, word representation, and model training.

A. *Exploratory Data Analysis (EDA)*

The dataset was already split into 80% training and 20% testing. In EDA, we checked class distribution for imbalance, analyzed text length, explored frequent words and n-grams, and identified noisy tokens like punctuation and numbers. These insights guided preprocessing, emphasizing text normalization and noise reduction.

B. *Preprocessing*

To ensure uniformity and improved feature extraction, the following preprocessing steps were applied to both training and test data:

- **Lowercasing** all text to remove case sensitivity.
- **Tokenization** to split documents into meaningful units (words).
- **Stopword removal** using standard NLP stopword lists.
- Special symbols, numbers, and extra whitespace were removed.

The cleaned text corpus was then used for feature representation.

C. *Word Representations*

Text was numerically represented using BoW, TF-IDF, GloVe, and Word2Vec skip-gram embeddings:

- **BoW** – Represented documents as unigram-based vectors, with dimensionality reduced by limiting the vocabulary size to a fixed maximum number of features.
- **TF-IDF** – Generated weighted unigram vectors that account for term frequency and inverse document frequency, also constrained by a maximum vocabulary size.
- **GloVe** – Utilized pre-trained 100-dimensional word embeddings to capture semantic similarity, with out-of-vocabulary (OOV) words initialized randomly.
- **Skip-gram** – Trained Word2Vec skip-gram embeddings using negative sampling to model contextual relationships between words.

D. *Models*

- **Machine Learning(ML) Models:** Logistic Regression, Naive Bayes, and Random Forest with tuned hyperparameters.
- **Neural Network(NN) Models:** DNN, SimpleRNN, GRU, LSTM, and their bidirectional variants. Tuning involved hidden units, dropout rates, optimizers, and learning rates. Early stopping was used to prevent overfitting.

E. *Hyperparameters*

Hyperparameters for each model were carefully selected based on validation performance, training efficiency, and overfitting prevention.

1) For BoW (Bag of Words):

- **BoW + Random Forest :**with 100 estimators, max depth 25, min samples split 5, and min samples leaf 2 was chosen to balance accuracy, training time, and prevent overfitting.
- **BoW + Logistic Regression:**with max_iter=2000 and n_jobs=-1 was used to ensure convergence in high-dimensional BoW, with L2 regularization preventing overfitting.
- **BoW + Naive Bayes :**MultinomialNB with default settings was used as it works well on sparse count features, is fast, and requires no heavy tuning.
- **BoW + Deep Neural Network :**A small DNN (64 to Dropout 0.3 to 32 to softmax) with Adam optimized training was used to reduce overfitting and improve generalization.

*2) **For TF-IDF** :*

- **TF-IDF + Random Forest:** with 100 estimators, max depth 25, min samples split 5, and min samples leaf 2 was chosen to balance depth and tree quantity for stable predictions while preventing very small splits.

- **TF-IDF + Logistic Regression:** with max_iter=2000 and n_jobs=-1 was used to ensure convergence on high-dimensional TF-IDF vectors, with parallel jobs speeding up training.
- **TF-IDF + Naive Bayes:** MultinomialNB with default settings was applied as it works well with TF-IDF counts and requires no heavy tuning.
- **TF-IDF + Deep Neural Network:** A DNN (256 to $\rightarrow$ 128 $\rightarrow$ Dropout 0.2 to 64 to softmax) with Adam ; dropout prevented overfitting.

*3) **For Glove Embedding** :*

- GloVe + DNN: Embedding frozen, Flatten to Dense 128 to Dropout 0.3 to 64 to softmax; small DNN avoids overfitting, dropout added for regularization; epochs=8, batch=32.
- **GloVe + SimpleRNN:** Two RNN layers 64 to 32, relu, dropout=0.2, recurrent_dropout=0.2; smaller units prevent vanishing gradients and overfitting; epochs=7, batch=32.
- **GloVe + GRU:** GRU layers 64 to 32, relu, dropout=0.3 after last GRU; chosen for better long-term memory; epochs=10, batch=32.
- **GloVe + LSTM:** LSTM layers 128 to 64, dropout=0.3; larger first layer for complex patterns, smaller second layer for dimensionality reduction; epochs=8, batch=64.
- **GloVe + Bi-SimpleRNN:** Bidirectional RNN 128 to 64, relu, dropout=0.3; captures context both directions; epochs=8, batch=64.
- **GloVe + Bi-GRU:** Bidirectional GRU 128 to 64, relu, dropout=0.3; faster than LSTM, bidirectional for context; epochs=10, batch=64.
- **GloVe + Bi-LSTM:** Bidirectional LSTM 64 to 64, dropout=0.3; balanced units to capture sequence info without overfitting; epochs=10, batch=64.

*4) For **Skip-Gram** :*

- **Skip-gram + DNN:** Dense layers 128 to 64 to softmax, dropout=0.3; simple feedforward network, dropout prevents overfitting, units balance learning and overfitting risk; epochs=20, batch=32.
- **Skip-gram + SimpleRNN:** Dense layers 64 to 32 to softmax; sequence info lost in averaged embeddings, dense layers used instead; epochs=10, batch=128.
- **Skip-gram + GRU:** Dense layers 64 to 32 to softmax; GRU used as a proxy for sequence modeling, smaller units due to averaged embeddings; epochs=10, batch=128.
- **Skip-gram + LSTM:** Dense layers 64 to 32 to softmax; smaller layers since averaged embeddings have limited sequence info; epochs=10, batch=128.
- **Skip-gram + Bidirectional SimpleRNN:** Dense layers 64 to 32 to softmax; bidirectional concept applied despite minimal sequence info; epochs=10, batch=128.
- **Skip-gram + Bidirectional GRU:** Dense layers 128 to 64 to Dropout 0.3 to 32 to softmax; larger layers allow bidirectional flow for richer representation; epochs=10, batch=64.
- **Skip-gram + Bidirectional LSTM:** Dense layers 64 to 32 to softmax; smaller units sufficient for averaged embeddings, bidirectional helps capture residual context; epochs=10, batch=256.

III. Results

This section presents the performance of all models evaluated in the project. Models were trained using different word representations (BoW, TF-IDF, GloVe, Skip-gram) combined with machine learning and neural network architectures. Performance was measured using Accuracy, F1-Macro, and F1-Weighted scores.

A. Overall Performance

TABLE I
Performance of 22 Models

Model	Type	Accuracy	F1-Macro	F1-Weighted
TFIDF + RandomForest	ML	0.4893	0.4968	0.4968
TFIDF + LogisticRegression	ML	0.5943	0.5904	0.5904
TFIDF + NaiveBayes	ML	0.5784	0.5735	0.5735
BoW + RandomForest	ML	0.5077	0.5104	0.5104
BoW + LogisticRegression	ML	0.6207	0.6163	0.6163
BoW + NaiveBayes	ML	0.6165	0.6125	0.6125
Skipgram + DNN	NN	0.6955	0.6874	0.6874
Skipgram + SimpleRNN	NN	0.6910	0.6847	0.6847
Skipgram + GRU	NN	0.6897	0.6841	0.6841
Skipgram + LSTM	NN	0.6907	0.6854	0.6854
Skipgram + BiSimpleRNN	NN	0.6914	0.6857	0.6857
Skipgram + BiLSTM	NN	0.6903	0.6845	0.6845
Skipgram + BiGRU	NN	0.6929	0.6878	0.6878
GloVe + DNN	NN	0.6396	0.6341	0.6341
GloVe + SimpleRNN	NN	0.2373	0.1522	0.1522
GloVe + GRU	NN	0.7082	0.7032	0.7032
GloVe + LSTM	NN	0.7138	0.7081	0.7081
GloVe + BiSimpleRNN	NN	0.6947	0.6898	0.6898
GloVe + BiGRU	NN	0.6849	0.6828	0.6828
GloVe + BiLSTM	NN	0.6981	0.6943	0.6943
TFIDF + DNN	NN	0.5768	0.5715	0.5715
BoW + DNN	NN	0.6057	0.6010	0.6010

B. Model Performance Plots

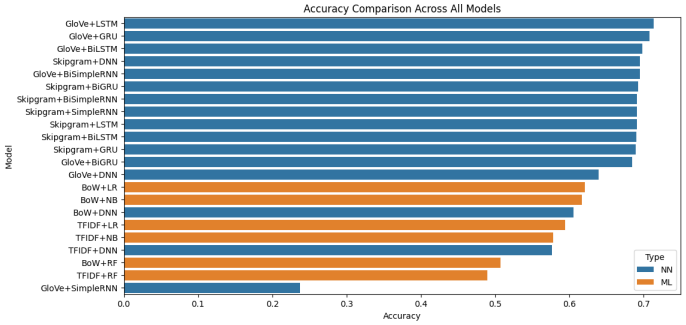

Fig. 1. Accuracy comparison across models.

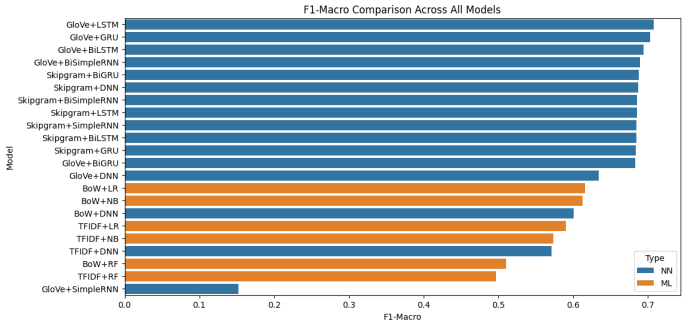

Fig. 2. F1-Macro score comparison across models.

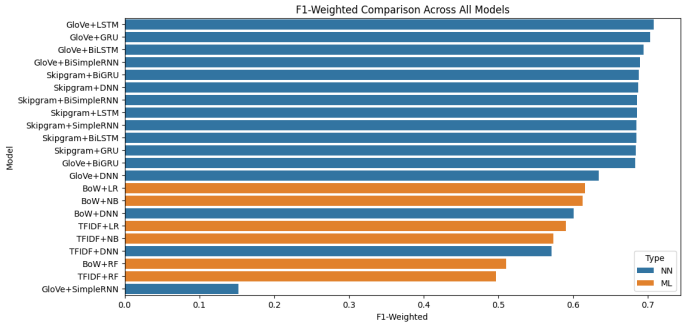

Fig. 3. F1-Weighted score comparison across models.

C. Best and Worst Models

TABLE II
BEST AND WORST PERFORMING MODELS

Category	Model	Accuracy	F1-Macro	F1-Weighted
Best ML Model	BoW + Logistic Regression	0.6207	0.6163	0.6163
Worst ML Model	TFIDF + RandomForest	0.4893	0.4968	0.4968
Best NN Model	GloVe + LSTM	0.7138	0.7081	0.7081
Worst NN Model	GloVe + SimpleRNN	0.2373	0.1522	0.1522
Best Overall Model	GloVe + LSTM	0.7138	0.7081	0.7081
Worst Overall Model	GloVe + SimpleRNN	0.2373	0.1522	0.1522

D. Best/Worst Model Accuracy Plots

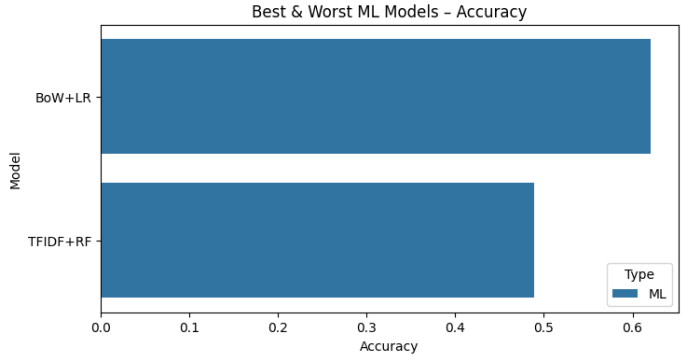

Fig. 4. ML Model Accuracy: Best and Worst

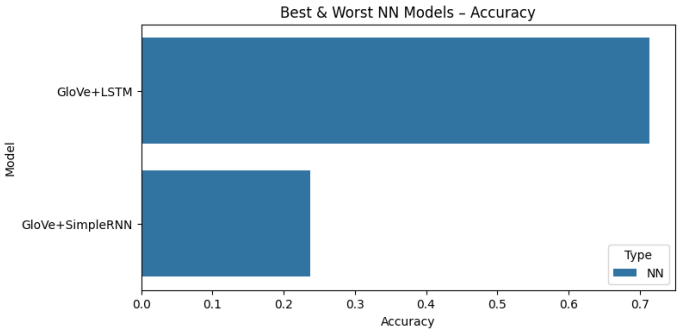

Fig. 5. NN Model Accuracy: Best and Worst

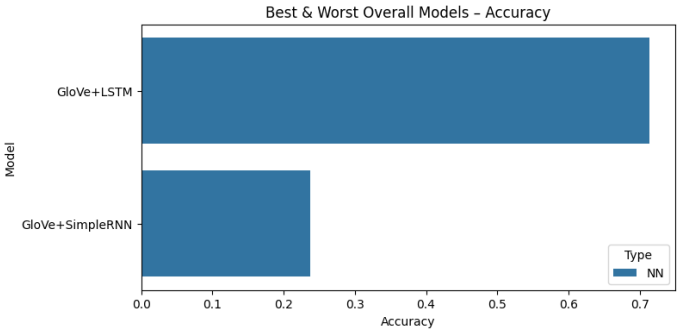

Fig. 6. Overall Model Accuracy: Best and Worst

E. Comparative Insights

Sequential models with pre-trained embeddings (GloVe) consistently outperformed frequency-based features. Logistic Regression was the strongest ML baseline, surpassing both Random Forest and Naive Bayes. Among neural networks, GloVe+LSTM demonstrated superior ability to capture long-range dependencies, while GloVe+SimpleRNN struggled due to vanishing gradients.

IV. CONCLUSION

This project demonstrates the impact of different word representations on multi-class text classification. Pre-trained embeddings combined with sequential models such as LSTM and GRU outperformed traditional frequency-based features. The best-performing model, GloVe + LSTM, achieved an accuracy of 71.38% with a macro F1 score of 0.7081. Key challenges included class imbalance and the high computational cost of training deep models. Future work could explore transformer-based architectures, methods for addressing class imbalance, and the use of domain-specific embeddings to further improve performance.

REFERENCES

[1] CSE440 Project Dataset (Summer 2025). Available: https://drive.google.com/drive/folders/1FMACf215tf3EuqmRDP-XEDnahyT1kea_

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv preprint arXiv:1301.3781*, 2013.

[3] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.
[4] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[5] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," *arXiv preprint arXiv:1406.1078*, 2014.
[6] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997.
[7] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.
[8] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
[9] R. Johnson and T. Zhang, "Effective Use of Word Order for Text Categorization with Convolutional Neural Networks," *arXiv preprint arXiv:1412.1058*, 2014.
[10] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[11] M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *arXiv preprint arXiv:1603.04467*, 2016.
[12] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. Conf. Neural Information Processing Systems (NeurIPS)*, 2019, pp. 8024–8035.
[13] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.