Bag of Words Document Classification using Feedforward Neural Network and Recurrent Neural Network

ANDREAS PEDERSEN, University of Stavanger, Norway
MARIUS HAUGE HÅLAND, University of Stavanger, Norway
SIMON LUNDGREN, University of Stavanger, Norway
SONDRE LYNGSTAD, University of Stavanger, Norway

This paper explores document classification using Bag-of-Words (BoW) representations and pre-trained word embeddings in combination with two neural network architectures: Feedforward Neural Networks (FFNNs) and Recurrent Neural Networks (RNNs). We evaluate how each model performs the task of classifying documents into predefined categories based on their abstracts from the ArXiv100 dataset. Our experiments include BoW methods such as CountVectorizer and TF-IDF, as well as embedding-based approaches using Word2Vec, GloVe, and FastText. We compare performance across network configurations and pooling strategies, analyzing accuracy, precision, recall, and F1-score. The results highlight the strengths and limitations of FFNNs and RNNs when applied to various text representation techniques.

GitHub Repository

CCS Concepts: • **Computing methodologies** → **Supervised learning**; *Natural language processing*; • **Information systems** → *Document representation.*

ACM Reference Format:
Andreas Pedersen, Marius Hauge Håland, Simon Lundgren, and Sondre Lyngstad. 2026. Bag of Words Document Classification using Feedforward Neural Network and Recurrent Neural Network. 1, 1 (February 2026), 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Text classification is critical in organizing, searching, and analyzing large-scale document collections. With the ever-increasing volume of scientific publications, automatic classification of research papers is essential for efficient information retrieval and categorization [15].

In this project, we address the problem of classifying scientific abstracts from the ArXiv100 dataset [4] into one of ten predefined research fields. The abstracts contain complex and domain-specific language, making this a challenging natural language processing (NLP) task.

We explore multiple feature extraction methods, including Bag-of-Words and pre-trained word embeddings, to solve this. We implement feedforward neural networks (FFNN) on static representations and recurrent neural networks (RNN) on sequence data for classification. Both frozen and fine-tuned embeddings are considered.

Our goal is to evaluate and compare these methods to determine which combination yields the best performance for scientific abstract classification.

Authors' addresses: Andreas Pedersen, University of Stavanger, Stavanger, Norway, andreas.pedersen@stud.uis.no; Marius Hauge Håland, University of Stavanger, Stavanger, Norway, mah.haland@stud.uis.no; Simon Lundgren, University of Stavanger, Stavanger, Norway, simon.lundgren@gmail.com; Sondre Lyngstad, University of Stavanger, Stavanger, Norway, srlyng@gmail.com.

2 BACKGROUND

This section describes the different methods for representing text, the models used for classification, and the training strategy employed in this project.

2.1 Vectorizers

2.1.1 *CountVectorizer.*

BoW method that transforms a collection of text documents into a matrix of token counts [14]. Given a corpus of m documents:

$$D = \{d_1, d_2, \ldots, d_m\}$$

and a vocabulary of n unique words extracted from the corpus:

$$V = \{w_1, w_2, \ldots, w_n\}$$

each document d_i is represented as a vector $\mathbf{x}_i \in \mathbb{R}^n$, where the j-th entry corresponds to the count of word w_j in document d_i:

$$\mathbf{x}_i = [f_{i1}, f_{i2}, \ldots, f_{in}]$$

where

$$f_{ij} = \text{count}(w_j \in d_i)$$

Stacking these vectors results in a document-term matrix where $X \in \mathbb{R}^{m \times n}$:

$$X = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mn} \end{bmatrix}$$

2.1.2 *TFIDFVectorizer.*

TF-IDF is another BoW method that transforms a collection of text documents into numerical vectors using the TF-IDF method. TF, short for *term frequency*, is the part that counts how often a word appears in each document [9].

$$\text{TF}(t, d) = \frac{f(t, d)}{|d|} \tag{1}$$

Where $f(t, d)$ is the frequency of term t in document d and $|d|$ is the total number of words in document d. IDF, short for *inverse document frequency*, is the part that calculates the uniqueness/significance of a word compared to all of the documents. Words that can be found mostly in one document, compared to all of the documents, get a high score. Words that can be found across all the documents get a low score.

$$IDF(t) = log(\frac{\mathrm{N}}{1 + \mathrm{df(t)}}) \tag{2}$$

Where N is the total number of documents and *df(t)* is the number of documents containing the word *t*. By combining TF and IDF, you give higher scores to words that can be found more often in one document but rarely across all the documents.

$$TFIDF(t, d) = TF(t, d) \cdot IDF(t) \tag{3}$$

This score reflects how unique and important a word is for a specific document.

2.2 Word Embeddings

Word embeddings are dense vector representations of words, where the geometric relationship between vectors captures both semantic and syntactic information. In this project, three embedding methods are utilized: *Word2Vec*, *GloVe*, and *FastText*.

2.2.1 *Word2Vec.*

A predictive embedding model introduced by Mikolov et al. (2013) that learns dense vector representations of words by maximizing the probability of word co-occurrences within a fixed-size context window [11]. It offers two main architectures: **Continuous Bag of Words (CBOW)**, which predicts a target word from its surrounding context, and **Skip-gram**, which does the opposite — predicting context words from a target word. Training is performed using a shallow neural network optimized via techniques like *negative sampling* or *hierarchical softmax*, allowing for efficient learning even on large corpora. Word2Vec embeddings capture rich syntactic and semantic relationships and support vector arithmetic. However, it treats words as atomic units, which limits its ability to handle out-of-vocabulary (OOV) words or morphologically rich languages.

2.2.2 *GloVe.*

Glove (Global Vectors for Word Representation) is a count-based embedding model developed by Pennington et al. (2014) at Stanford [7]. Unlike Word2Vec, which focuses on local context windows, GloVe leverages global word-word co-occurrence statistics across the entire corpus. It constructs a large matrix where each entry represents how frequently two words co-occur, and then learns word vectors by factorizing this matrix using a weighted least squares objective. The goal is to find vector representations such that their dot product approximates the logarithm of their co-occurrence frequency. GloVe captures both syntactic and semantic regularities and produces embeddings that excel in analogy reasoning. However, similar to Word2Vec, it cannot generate embeddings for unseen words, as it relies on precomputed co-occurrence counts.

2.2.3 *FastText.*

FastText was introduced by Bojanowski et al. (2016) at Facebook AI Research. Fasttext extends the Word2Vec model by incorporating subword information [13]. Rather than learning a single vector per word, FastText represents each word as a bag of character-level n-grams (e.g., “running” would include “run”, “unn”, “nni”, etc.). The final word embedding is computed as the sum of its subword embeddings. This approach enables the model to generate vectors for rare or even unseen words by composing them from known subword units. As a result, FastText is particularly useful in low-resource or morphologically complex languages and applications with frequent out-of-vocabulary terms. It retains the efficiency of Word2Vec while improving generalization and robustness.

2.3 Neural Network Models

In this section, we look at the theoretical foundations of feedforward neural networks and recurrent neural networks, which are both utilized in our project.

2.3.1 *Feedforward Neural Network - FFNN.*

A Feedforward Neural Network is one of the most fundamental types of artificial neural networks used in machine learning ([6], page 164). It is called “feedforward” because the information flows in a single direction, from the input layer, through one or more hidden layers, and finally to the output layer. It does this without any loops or cycles. An illustration of such a neural network is shown in Fig. 1.

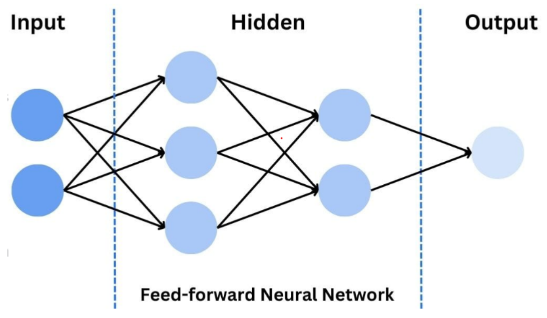

Fig. 1. Illustration of a Feedforward Neural Network [3]

The architecture of an FFNN begins with an input layer that receives the input features from the dataset. Each input neuron corresponds to one element in the input vector, such as a specific word in a vocabulary when using bag-of-words features. Following the input layer are one or more hidden layers. Each neuron in a hidden layer performs a computation involving a weighted sum of its inputs, adds a bias term, and applies a non-linear activation function. Examples of such activation functions are *ReLU* or *Sigmoid*. These hidden layers allow the network to learn complex relationships in the data. How this model is trained and the parameters tuned are discussed more in Sec. 2.4.

The final layer is the output layer, which produces the prediction of the network. The output can be calculated using different types of *softmax* functions. These functions convert the raw scores into probabilities, and thus, the class with the highest probability is the one chosen.

2.3.2 *Recurrent Neural Network - RNN.*

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to model sequential data. Unlike traditional feedforward neural networks (FFNNs), where information flows in only one direction, RNNs use loops to maintain information across time steps, allowing the network to handle patterns that evolve [10]. This makes them particularly effective for tasks involving sequences, such as text, where the order of words matters. However, basic RNNs often struggle with the vanishing gradient problem, which limits their ability to capture long-range dependencies. More advanced architectures, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), were developed to address these challenges by controlling the flow of information [5]. An illustration of a recurrent neural network is shown in Fig. 2.

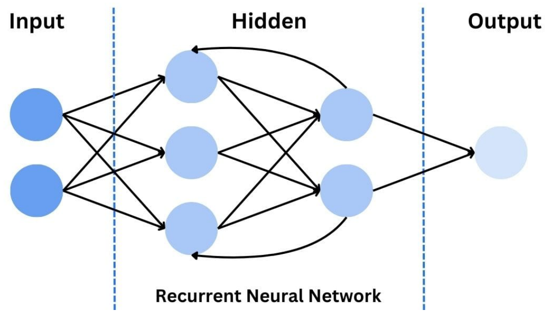

Fig. 2. Illustration of a Recurrent Neural Network [3]

The architecture of an RNN also begins with an input layer that receives the input features from the dataset, such as in Sec. 2.3.1, but unlike an FFNN, it is specifically designed to handle sequences of data. Following the input layer is a hidden layer that, at each time step, takes both the current input and the previous hidden state as input. Each neuron in a hidden layer performs the same computations as they do in FFNN in Sec. 2.3.1. This structure allows the network to retain information across time steps and learn temporal patterns in the data. The output layer produces the prediction of the network, often based on the final hidden state or all hidden states combined.

2.4 Training Strategy

Training of the neural network models relies on the **cross-entropy loss** function, a standard choice for multi-class classification tasks [8]. Cross-entropy quantifies the discrepancy between the model's predicted probability distribution and the actual class label, effectively measuring how confident and correct the model's predictions are.

2.4.1 *Cross-Entropy Loss.*

Given an input sample x, the model produces a vector of raw scores (logits) $\mathbf{z} = f(x) \in \mathbb{R}^C$, where C denotes the number of output classes. These logits are transformed into class probabilities via the *softmax* function in Equation 4.

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{4}$$

If the true class label is $y \in \{1, \dots, C\}$, the cross-entropy loss for a single sample is defined as:

$$\mathcal{L}(x, y) = -\log_2(p_y)$$

This expression penalizes the model more heavily when it assigns a low probability to the correct class. During training, the average loss over a mini-batch of N samples is computed as:

$$\mathcal{L}_{\text{batch}} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(x^{(n)}, y^{(n)})$$

This batch-wise average loss provides a smooth training signal for optimization and is used to guide the updates of model parameters via backpropagation.

2.4.2 *Training and Validation Loss.*

Throughout training, the model computes the average cross-entropy loss over training batches that is used to update model parameters via gradient descent and validation batches that are evaluated without updating model weights, serving solely to monitor generalization performance.

Tracking both training and validation loss across epochs enables the detection of underfitting and overfitting. A decreasing training loss paired with an increasing validation loss typically indicates that the model is starting to overfit, i.e., memorizing the training data instead of learning generalizable patterns.

2.5 Evaluation metrics

To assess the performance of our classification models, we use standard evaluation metrics commonly applied in multi-class classification tasks. These metrics provide insights into how well the model predicts the correct classes and how balanced its predictions are across different categories. In this project, we focus on four key metrics: accuracy, precision, recall, and F1-score. In the following explanations of the four key metrics, we use the abbreviations: TP for true positives, TN for true negatives, FN for false negatives, and FP for false positives [1].

2.5.1 *Accuracy.*

Accuracy measures the proportion of correct predictions made by the model across all classes and is defined as:

$$\mathit{Accuracy} = \frac{TP + TN}{TP + TN + FN + FP} \tag{5}$$

Accuracy is easy to interpret and provides a general sense of model performance. However, it can be misleading in cases of class imbalance, where a model may achieve high accuracy by simply predicting the majority class.

2.5.2 *Precision.*

Precision measures the proportion of correctly predicted positive instances out of all instances predicted as positive, and is defined as:

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

High precision indicates that the model makes few false positive errors. However, precision does not account for false negatives, and may be misleading if used alone in situations where missing positive cases is costly.

2.5.3 *Recall.*

Recall measures the proportion of actual positive instances that were correctly identified by the model, and is defined as:

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

High recall means the model successfully detects most positive cases. However, it does not consider how many false positives are made, and high recall alone may come at the cost of lower precision.

2.5.4 *F1-Score.*

F1-score is the harmonic mean of precision and recall, providing a balanced measure that accounts for both false positives and false negatives. It is defined as:

$$F1\_score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{8}$$

A high F1-score indicates a good balance between precision and recall. It is especially useful in scenarios with class imbalance or when both false positives and false negatives carry significant consequences.

2.6 Pooling Methods

Pooling methods are used to aggregate variable-length sequences of word embeddings into fixed-size vectors, making them suitable as input for neural network models. Different pooling strategies emphasize different aspects of the data.

2.6.1 *Max Pooling.*

Selects the maximum value for each feature across all word embeddings in the sequence. It captures the most dominant features, highlighting the strongest signal present in any word. This method can be useful when keywords are strong indicators of the document's class, regardless of their position [2].

2.6.2 *Mean Pooling.*

Computes the mean value for each feature across all word embeddings. This approach summarizes the overall semantic content of the document and reduces sensitivity to the length of the input. It provides a more balanced representation by considering all words equally [2].

2.6.3 *Sum Pooling.*

Calculates the element-wise sum across all word embeddings. Like average pooling, it aggregates information across the entire document, but without normalizing by sequence length. Sum pooling can highlight documents with more frequent or stronger feature activations, but it may be more sensitive to document length [12].

3 METHODOLOGY

3.1 Overview

The goal is to systematically evaluate different vectorization techniques, embedding strategies, and model architectures for document classification on a scientific abstracts dataset. Our approach and implementation consist of three main stages:

(1) **Data preprocessing**: *Data cleaning* 3.2, *Vectorization Techniques* 3.3, and *Word Embeddings* 3.4
(2) **Model architectures** 3.6
(3) **Training and evaluation**: *Procedure* 3.7 and *Metrics* 3.8

3.2 Data cleaning

The input data consists of abstracts extracted from the arXiv corpus. Initial preprocessing steps include:

- Lowercasing all text.
- Removing stop words using NLTK's English stopword list.
- Tokenization into word units.
- Optional cleaning, such as removal of special characters.

This preprocessing pipeline ensures that the vocabulary is clean, consistent, and free from noisy tokens.

3.3 Vectorization Techniques

The two vectorization techniques explained in Sec. 2.1 were implemented by utilizing the `scikit-learn` package for machine learning in Python.

3.4 Word Embeddings

In addition to sparse BoW representations, we explored dense word embeddings. Both pre-trained and custom embeddings were evaluated:

3.4.1 *Pre-trained Embeddings.*

- **Word2Vec**: GoogleNews-vectors-negative300.
- **GloVe**: GloVe 6B 100d / 300d.
- **FastText**: Crawl-based with subword information.

Pre-trained embeddings were either kept static (frozen) or fine-tuned during training.

3.4.2 *Custom Embeddings.*

To match the domain-specific nature of scientific abstracts, we trained custom Word2Vec and FastText models on the dataset using the Python package `gensim`. Training parameters included:

- `vector_size`: 300
- `window`: 5
- `min_count`: 3
- `epochs`: 10

Custom embeddings allowed the model to capture dataset-specific word usage more effectively.

3.5 Embedding Pooling Strategies

For static embeddings (non-fine-tuned), we tested multiple pooling strategies to create fixed-length input vectors:

- Mean pooling
- Min pooling
- Max pooling
- Sum pooling

These strategies aggregate word vectors at the document level for compatibility with feedforward architectures.

3.6 Model Architectures

3.6.1 *Feedforward Neural Networks (FFNN).*

The FFNN models take either BoW vectors or pooled embedding vectors as input. The architecture consists of:

- An input layer matching the dimensionality of the feature vector.
- Two or three hidden layers (e.g., 256–128–64 neurons) with activation functions such as *ReLU* or *Sigmoid*.
- Dropout regularization after each hidden layer to prevent overfitting.
- An output layer corresponding to the number of classes, trained using cross-entropy loss.

When fine-tuning embeddings, an embedding layer was appended to the FFNN to allow back-propagation into the word vectors.

3.6.2 *Recurrent Neural Networks (RNN).*

For sequence modeling, the RNN variants LSTM and GRU were tested. We performed mainly three experiments for the two RNNs:

- Exploring which pooling method (sum, max, or mean) yields the best performance.
- Exploring which type of pre-trained word embedding (Word2Vec, FastText, or GloVe) yields the best performance.
- Exploring the effect of fine-tuning the pre-trained word embeddings.

Key settings used and kept consistent when testing the pooling methods for the GRU and LSTM model:

- Three stacked hidden layers with sizes 256, 128, and 64.
- Vocabulary size was limited to 20,000 tokens, and the maximum sequence length was set to 300.
- Pre-trained GloVe embeddings with 100 dimensions. Kept frozen during training.

The pooling method yielding the highest scores was used for further testing the different embedding types.

3.7 Training Procedure

- Loss Function: Cross-Entropy Loss.
- Batch Size: 16 (RNN) and 64 (FFNN).
- Early stopping based on validation loss to avoid overfitting.
- Model selection based on the best validation performance.

Training metrics (training loss, validation loss) were logged at each epoch and exported to CSV for later analysis.

3.8 Evaluation Metrics

Model performance was evaluated using the four metrics explained in Sec. 2.5. These metrics were computed on a held-out testing set, separate from the training data, to ensure a fair and unbiased estimate of generalization performance. All model predictions were also saved together with their corresponding true labels. This allows for the creation of a *confusion matrix* to visualize the distribution of true versus predicted labels, offering a more detailed view of class-wise performance and misclassification patterns.

4 RESULTS

This chapter presents the experimental results obtained from training and evaluating the FFNN and RNN models. To ensure sufficient computational power for training and evaluation, the experiments were primarily performed on the University of Stavanger UNIX *gorina6* server. The findings are discussed in relation to the effectiveness of different feature representations and model architectures for multi-class text classification.

4.1 Feed Forward Neural Network - FFNN

For the FFNN, we use static loaded embeddings, as well as using the BoW methods CountVectorizer and TF-IDF. Firstly, we show the results for BoW methods.

4.1.1 *CountVectorizer & TF-IDF.*

To represent the text data, we used *CountVectorizer* and *TF-IDF* as input to a Feedforward Neural Network (FFNN). The results are presented in Table 1.

Table 1. Evaluation metrics using FFNN and the two BoW methods.

BoW-method	**CountVectorizer**	**TF_IDF**
Accuracy (%)	**82.63**	81.17
Precision (%)	**82.52**	81.31
Recall (%)	**82.63**	81.16
F1_score (%)	**82.54**	81.22

CountVectorizer achieved slightly better performance across all metrics, including accuracy, precision, recall, and F1-score. Although the differences were small, the results suggest that simple word frequency was slightly more effective than TF-IDF weighting for this task. This may indicate that common words in the dataset carried useful information for classification, and that down-weighting them with TF-IDF reduced performance slightly. Overall, both methods performed well, but *CountVectorizer* proved to be marginally more suitable in this case.

4.1.2 *Static embedding.*

To evaluate the impact of different word embeddings and pooling strategies on classification performance, we trained a feedforward neural network using the three types of pre-trained static embeddings, GloVe, FastText, and Word2Vec. For each embedding type, we applied several vector pooling methods to transform variable-length token sequences into fixed-size input vectors. These pooled representations were then used to train and evaluate a classifier on the dataset.

Table 2 shows the classification performance across GloVe embedding and pooling combinations using metrics such as accuracy, precision, recall, and F1-score.

Table 2. Evaluation metrics for different pooling strategies using FFNN and GloVe Static embedding

Pooling	Max	Mean	Sum
Accuracy (%)	60.37	75.98	**79.57**
Precision (%)	59.90	75.75	**79.46**
Recall (%)	60.36	75.98	**79.57**
F1_score (%)	59.67	75.76	**79.44**

Based on these results, we observe that the pooling method *sum* outperformed other embedding types. To streamline training and focus on the most promising approaches, we excluded these underperforming pooling methods from further evaluation with custom-trained embeddings. Figure 3 shows the confusion matrix for the FFNN model using static GloVe embeddings with sum pooling.

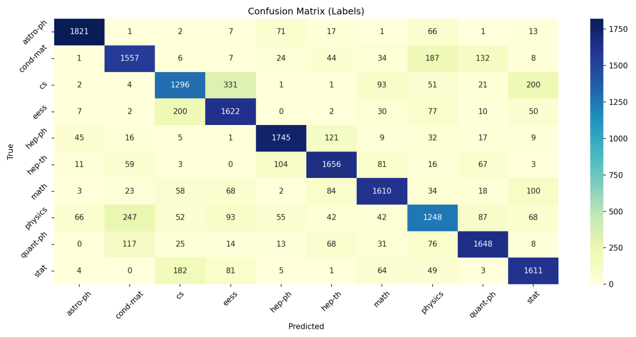

Fig. 3. Confusion Matrix for FFNN.

From Figure 3 we observe that, as the accuracy score from table 2 implies, the model mostly assigns the correct label. The matrix can also tell us that there are specific labels the model mistakenly guesses repeatedly. This includes *physics* and *condensed matter*. This could be due to the similarity in the field-specific vocabulary.

We then trained our custom embeddings on the dataset using Gensim's implementations of Word2Vec and FastText. The same neural network architecture was used to compare these custom embeddings against their pre-trained counterparts. Results for selected pooling methods are shown in Table 3.

Table 3. Evaluation metrics for different embedding strategies using an FFNN with custom-trained embedding and sum pooling

Embedding	Word2Vec	FastText
Accuracy (%)	**82.70**	77.82
Precision (%)	**82.64**	77.77
Recall (%)	**82.70**	77.81
F1_score (%)	**82.61**	77.68

The results demonstrate that custom-trained embeddings can yield competitive and even superior performance compared to large-scale pre-trained alternatives. Notably, custom Word2Vec embeddings were trained on just 1.6 million tokens from the dataset. While it may seem counterintuitive that small-scale embeddings outperform models trained on billions of words, such as GloVe or FastText, this outcome underscores the importance of domain relevance. Embeddings trained directly on domain-specific data are more likely to capture the nuances and specialized vocabulary of the task, leading to better generalization. In contrast, general-purpose pre-trained embeddings, which are also frozen during training, cannot adapt to the target task and may under-represent domain-specific terms. These findings suggest that even modestly sized, in-domain corpora can provide significant advantages when the classification task closely aligns with the source of the training data.

Table 4. Evaluation metrics for different embedding strategies using FFNN and **fine-tuned embeddings**.

Embedding	Word2Vec	FastText
Accuracy (%)	78.66	**79.08**
Precision (%)	78.68	**79.16**
Recall (%)	78.66	**79.08**
F1_score (%)	78.63	**79.08**

After experimenting with the fine-tuned embedding, the results show weaker performance compared with the static embedding.

4.2 Recurrent neural network - RNN

For RNN, we explored both LSTM and GRU. We first present the results obtained using GRU as the neural network.

4.2.1 GRU.

The results of the different pooling methods are summarized in Table 5.

Table 5. Evaluation metrics for different pooling strategies using a GRU-based RNN. Max pooling demonstrates slightly better performance compared to average and sum pooling.

Pooling	Max	Mean	Sum
Accuracy (%)	**82.16**	81.40	81.79
Precision (%)	**82.25**	82.67	81.93
Recall (%)	**82.15**	81.39	81.79
F1_score (%)	**82.14**	81.70	81.82

Since max pooling yielded the best performance, it was selected as the fixed pooling strategy for all subsequent experiments involving the GRU-based RNN. Using this setup, we evaluated different static embedding strategies, specifically GloVe, Word2Vec, and FastText, and investigated the effect of freezing versus fine-tuning the embeddings during training. The evaluation results for the different pooling strategies trained from scratch, without the use of pre-trained embeddings, are presented in Table 6. Table 7 reports the performance obtained with various pre-trained embedding methods without fine-tuning, whereas Table 8 presents the results achieved when the embeddings are fine-tuned during training.

Table 6. Evaluation metrics for the model trained from scratch without pre-trained embeddings. Showing competitive results.

Metric	Training from scratch
Accuracy (%)	**82.72**
Precision (%)	**82.70**
Recall (%)	**82.70**
F1_score (%)	**82.26**

Table 7. Evaluation metrics for different embedding strategies using a GRU-based RNN with **frozen embeddings**.

Embedding	GloVe	Word2Vec	FastText
Accuracy (%)	82.16	**82.73**	82.20
Precision (%)	82.25	**83.00**	82.24
Recall (%)	82.15	**82.71**	82.21
F1_score (%)	82.14	**82.74**	82.14

Table 8. Evaluation metrics for different embedding strategies using a GRU-based RNN with **fine-tuned embeddings**.

Embedding	GloVe	Word2Vec	FastText
Accuracy (%)	**84.14**	82.27	79.83
Precision (%)	**84.03**	82.22	79.76
Recall (%)	**84.14**	82.27	79.81
F1_score (%)	**84.00**	82.19	79.72

From the results presented, several important observations can be made. When using frozen embeddings, Word2Vec provided the best performance across all evaluation metrics, slightly outperforming both GloVe and FastText. Notably, training the model from scratch without any pre-trained embeddings (Table 6) achieved competitive results, demonstrating that the model was able to learn useful representations even without external semantic knowledge.

However, when fine-tuning of the embeddings was allowed during training, GloVe emerged as the clear winner, achieving the highest accuracy, precision, recall, and F1 score as seen in Table 8. Fine-tuning the embeddings allowed the model to adapt the representations specifically to the task at hand, leading to a significant boost in performance, especially for GloVe. In contrast, Word2Vec and FastText did not benefit as much from fine-tuning, and FastText in particular showed a noticeable drop in performance.

Overall, these results highlight that, while pre-trained embeddings are helpful, the ability to fine-tune them can make a substantial difference. Furthermore, the choice of embedding source (GloVe, Word2Vec, FastText) interacts with whether the embeddings should be frozen or fine-tuned, and this choice should be carefully considered based on the task and available computational resources.

In Figure 4, the confusion matrix for the simulation using GloVe embedding, fine-tuning, and max pooling is presented.

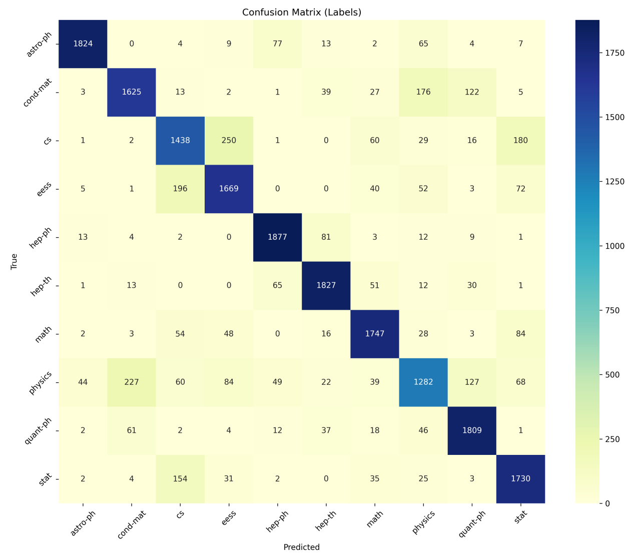

Fig. 4. Confusion Matrix for RNN (GRU).

From the confusion matrix, one can observe that the most common prediction error the model made was predicting *cs* (computer science), when the correct abstract was *eess* (electrical engineering and systems science), and vice versa. The second most common prediction error the model made was predicting *mat* (mathematics) when the true label was *physics*, and vice versa. The model achieved its highest accuracy when predicting the *hep-ph* (High Energy Physics) category. We believe the high/low prediction accuracy for the different subjects is due to the writing styles of the abstracts. Subjects with a lower prediction percentage have a less distinctive writing style, and thus are harder to separate from the other subjects.

4.2.2 LSTM.

Similar to the GRU experiments, the LSTM model was evaluated using different pooling strategies to determine which method yielded the best performance. The results for the different pooling strategies, using the LSTM model, are summarized in Table 9.

Table 9. Evaluation metrics for different pooling strategies using an LSTM-based RNN.

Pooling	Max	Mean	Sum
Accuracy (%)	82.16	82.55	**82.80**
Precision (%)	82.25	82.56	**82.89**
Recall (%)	82.16	82.52	**82.79**
F1_score (%)	81.87	82.37	**82.82**

Since *sum* pooling yielded the best performance, it was chosen as the fixed strategy for the remaining LSTM experiments. Tables 10, 11, and 12 present the results for the model trained from scratch without pre-trained embeddings, as well as the performance using GloVe, Word2Vec, and FastText embeddings with and without fine-tuning.

Table 10. Evaluation metrics for the model trained from scratch without pre-trained embeddings.

Metric	**Training from Scratch**
Accuracy (%)	**82.22**
Precision (%)	**82.24**
Recall (%)	**82.23**
F1_score (%)	**81.99**

Table 11. Evaluation metrics for different embedding strategies using an LSTM-based RNN with **frozen embeddings**.

Embedding	**GloVe**	**Word2Vec**	**FastText**
Accuracy (%)	82.80	**81.88**	81.64
Precision (%)	82.89	**82.15**	81.86
Recall (%)	82.79	**81.88**	82.62
F1_score (%)	82.82	**81.90**	81.52

Table 12. Evaluation metrics for different embedding strategies using an LSTM-based RNN with **fine-tuned embeddings**.

Embedding	**GloVe**	**Word2Vec**	**FastText**
Accuracy (%)	**83.88**	80.47	79.19
Precision (%)	**83.87**	80.78	79.15
Recall (%)	**83.87**	80.47	79.17
F1_score (%)	**83.81**	80.55	78.98

The LSTM experiments showed that *sum pooling* provided the best overall performance. Without fine-tuning, Word2Vec embeddings performed best, slightly outperforming GloVe, FastText, and random initialization. With fine-tuning enabled, GloVe embeddings led to the highest accuracy, highlighting the benefit of updating pre-trained embeddings during training. These results emphasize the importance of both the pooling strategy and the embedding fine-tuning for optimizing LSTM-based models.

5 CONCLUSION

This project explored document classification using Bag-of-Words (BoW) and pre-trained word embeddings combined with Feedforward Neural Networks (FFNN) and Recurrent Neural Networks (RNN), including LSTM and GRU architectures. Our results show that RNNs consistently achieved higher accuracy compared to FFNNs, particularly when using fine-tuned GloVe embeddings, which produced the best overall performance. However, this came at the cost of higher computational complexity and longer training times.

In the FFNN experiments, CountVectorizer slightly outperformed TF-IDF, suggesting that raw word frequency was more effective than down-weighted terms for this task. Additionally, sum pooling emerged as the most effective aggregation method for static embeddings within FFNNs, and custom-trained Word2Vec embeddings demonstrated strong performance, likely due to their domain specificity.

In the RNN experiments, max pooling performed best for GRU models, while sum pooling worked best with LSTM. Fine-tuning embeddings significantly improved performance for GloVe, highlighting the benefit of adapting pre-trained vectors to the target task.

Overall, the experiments underscore the importance of selecting appropriate embeddings, pooling strategies, and model architectures. While pre-trained models offer strong baselines, domain-adapted embeddings and careful tuning can yield better results in specialized text classification tasks.

6 ACKNOWLEDGMENTS

In this section, we acknowledge the use of AI in the development of this project. ChatGPT by OpenAI was used to assist in writing clarity, grammar correction, and to improve the overall structure. GitHub Copilot and ChatGPT were used for code debugging and code optimization. These tools were used strictly to support our development. All conceptual contributions, experimental design choices, and final decisions are our own.

7 CONTRIBUTION

All four group members contributed equally to this task. The work was carried out collaboratively, with shared responsibility across all aspects, including idea development, literature review, experimentation, coding, debugging, and documentation. The results presented in this report are the product of a collective effort, shaped through regular discussions, brainstorming sessions, and a commitment to shared learning and mutual support throughout the project.

REFERENCES

[1] 2022. What is Accuracy, Precision, Recall and F1 Score? https://www.labelf.ai/blog/what-is-accuracy-precision-recall-and-f1-score.

[2] DhanushKumar. 2023. MAX POOLING. https://medium.com/@danushidk507/max-pooling-ef545993b6e4.

[3] Chris Engelbert. 2024. Image Recognition with Neural Networks: A Beginner's Guide. https://www.simplyblock.io/blog/image-recognition-with-neural-networks/. Accessed: March 31, 2025.

[4] Ashkan Farhangi, Ning Sui, Nan Hua, Haiyan Bai, Arthur Huang, and Zhishan Guo. 2022. Protoformer: Embedding Prototypes for Transformers. In *Advances in Knowledge Discovery and Data Mining: 26th Pacific-Asia Conference, PAKDD 2022, Chengdu, China, May 16–19, 2022, Proceedings, Part I*. 447–458.

[5] Rui Fu, Zuo Zhang, and Li Li. 2016. Using LSTM and GRU neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. 324–328. https://doi.org/10.1109/YAC.2016.7804912

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[7] Christopher D. Manning Jeffery Pennington, Richard Socher. 2014. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/. Accessed: April 6, 2025.

[8] Aamna Kamran. 2024. Cross Entropy Loss Function in Machine Learning. https://metaschool.so/articles/cross-entropy-loss-function.

[9] Fatih Karabiber. [n. d.]. TF-IDF Term Frequency-Inverse Document Frequency LearnDataSci. https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/.

[10] Larry R Medsker, Lakhmi Jain, et al. 2001. Recurrent neural networks. *Design and Applications* 5, 64-67 (2001), 2.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781* (2013).

[12] Mohit Mishra. 2023. Pooling: A Key Concept in Machine Learning. https://mohitmishra786687.medium.com/pooling-a-key-concept-in-machine-learning-81c05dcbce98.

[13] Facebook AI Research. 2025. fastText English Word Vectors. https://fasttext.cc/docs/en/english-vectors.html. Accessed: April 2, 2025.

[14] The scikit-learn developers. 2025. CountVectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. Accessed: March 27, 2025.

[15] Fang Zhang and Shengli Wu. 2024. An Instance-based Plus Ensemble Learning Method for Classification of Scientific Papers. arXiv:2409.14237 [cs.DL] https://arxiv.org/abs/2409.14237