UNIVERSITE DE LIEGE
FACULTE DES SCIENCES APPLIQUEES

Biomedical Text Classification using LSTM, GRU and Bahdanau Attention

Mvomo Eto Wilfried

Thesis presented to obtain the degree of :
Master of Data Science and Engineering

Thesis supervisor :
Ashwin Ittoo

Academic year: **2024 - 2025**

Acknowledgments

I sincerely thank Jehovah, the Most Merciful and Full of Wisdom, for granting me the strength and perseverance necessary to pursue my studies and complete this thesis. His infinite mercy has been a constant source of guidance in all my achievements, and I am eternally grateful to Him. I express my deepest gratitude to Professor Ashwin Ittoo for his wise advice, encouragement, and unwavering support throughout this research. His supervision played a crucial role in consolidating my knowledge and the quality of this work. I also thank all the professors and assistants at the University of Liège for their teaching and expertise, which greatly contributed to the success of this thesis. I would also like to express my heartfelt appreciation to my parents for their constant support and for creating an environment conducive to my studies. Their encouragement has always been an essential source of motivation. Finally, I am grateful to my friends for their friendship, shared experiences, and mutual support, which have enriched both my academic and personal journey. The camaraderie and help I received made this adventure truly rewarding, and for that, I am deeply thankful.

Abstract

This research evaluates the performance of deep learning models—Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM), with Bahdanau attention added—for biomedical text classification using two datasets: the MSKCC dataset for binary classification of genetic mutations and the Medical Abstracts dataset for multi-class disease categorization. The experiments showed that GRU models generally offer better training efficiency and balanced accuracy than LSTM models, and that class weighting proved more effective than Synthetic Minority Over-sampling Technique (technique for oversampling minority classes) (SMOTE) in handling class imbalance. While few-shot learning remains challenging, models using Biomedical Natural Language Processing (BioMedNLP) contextual embeddings combined with attention mechanisms demonstrated promising generalization, particularly in low-resource scenarios. These findings support the development of more robust and equitable Natural Language Processing (NLP) systems for biomedical applications.

Contents

Acronyms

AI Artificial Intelligence. 3

BERT Bidirectional Encoder Representations from Transformers. 11

BioBERT BioBERT, a BERT variant for the biomedical domain. 13

BioMedNLP Biomedical Natural Language Processing. ii, 1, 4, 12, 32, 44

FSL Few-Shot Learning. 13, 14

GloVe Global Vectors for Word Representation. 1, 4, 9–11, 13, 16, 17, 32, 44

GRU Gated Recurrent Unit. ii, 1, 4, 6–9, 16, 27, 47

LSTM Long Short-Term Memory. ii, 1, 4, 6–8, 13, 16, 27, 47

NLP Natural Language Processing. ii, 1, 3, 13, 16

RNN Recurrent Neural Network. 4–7, 16

SMOTE Synthetic Minority Over-sampling Technique (technique for oversampling minority classes). ii, 14, 15, 21, 30, 32, 35, 42, 46–48

t-SNE t-distributed Stochastic Neighbor Embedding. 12, 13

Word2Vec Method for learning word vector representations developed by Mikolov et al.. 11, 17

List of Figures

List of Tables

Introduction

General Context

The automatic classification of biomedical texts is a vital task in NLP, with applications ranging from clinical decision support and literature mining to disease surveillance and personalized medicine.[1] However, the biomedical domain presents unique challenges: its language is highly specialized, context-dependent, and constantly evolving. These characteristics make it difficult for general-purpose NLP models to perform reliably, especially when dealing with rare diseases or limited annotated data.

Given the vast scope of biomedical literature, this thesis focuses on two representative datasets that capture different facets of the classification challenge. The first is the Memorial Sloan Kettering Cancer Center (MSKCC) dataset, which provides expert-annotated descriptions of genetic mutations for binary classification with significant class imbalance. The second is the Medical Abstracts Dataset, which spans five major disease categories—neoplasms, cardiovascular, digestive, nervous system, and general pathological conditions—supporting multi-class classification and broader generalization analysis. Together, these datasets enable a nuanced evaluation of model performance across both high- and low-resource disease categories.

To address the complexity of biomedical text classification, this research evaluates the performance of sequential deep learning models—GRU and LSTM, with Bahdanau attention added—combined with various embedding strategies, including Global Vectors for Word Representation (GloVe), Fasttext, and BioMedNLP contextual embeddings. The study is guided by three central research questions: *How does training time vary across different model architectures and embedding strategies, and what trade-offs exist between performance and computational efficiency? What is the impact of different class imbalance mitigation strategies, such as SMOTE versus class weighting, on model performance across both high- and low-resource disease categories? And to what extent do the best-performing models generalize to underrepresented disease categories, and can few-shot learning methods effectively improve classification in low-resource scenarios?*

By investigating these questions, the thesis aims to contribute to the development of robust, efficient, and equitable NLP systems tailored to the biomedical domain.

Case Study 1: Genetic Mutations and Precision Oncology

Genetic mutations refer to changes in the DNA (Deoxyribonucleic Acid) sequence of an organism. These alterations can occur naturally or be induced by environmental factors, and they play a central role in the development of various diseases, particularly cancer. In oncology, mutations in specific genes can lead to uncontrolled cell growth, tumor formation, and metastasis. Understanding and classifying these mutations is essential for identifying disease mechanisms and guiding personalized treatment strategies.[2]

The MSKCC dataset focuses on the classification of genetic mutations based on clinical textual evidence. This dataset is particularly relevant in the context of *oncology*, where classifying clinically actionable mutations enables oncologists to identify mutations responsive to targeted therapies, improving outcomes and reducing unnecessary interventions.[3]

[1] Some parts of the text in this research work have been reformulated with the assistance of ChatGPT-3 to improve grammatical clarity and readability.

[2] See: `https://www.genome.gov/genetics-glossary/Mutation`

[3] See: `https://www.cancer.gov/about-cancer/treatment/types/precision-medicine`

Moreover, the dataset reflects real-world clinical annotation practices, making it a valuable benchmark for developing NLP models that can assist in automating mutation classification—a traditionally labor-intensive and time-consuming task. By leveraging this dataset, the thesis contributes to the broader goal of *accelerating precision oncology*, where AI-driven tools support clinicians in interpreting genetic data and making informed treatment decisions.

Case Study 2: Disease Categorization in Epidemiology

Pathological diseases encompass a wide range of medical conditions that affect the structure and function of organs and tissues, including cancers, cardiovascular diseases, neurological disorders, and digestive system diseases. Accurate classification of these conditions is essential for diagnosis, treatment planning, and public health monitoring.

The Medical Abstracts Dataset comprises clinical summaries categorized into five major disease groups, making it representative of *public health and epidemiological research.* Disease classification is foundational to healthcare systems, with standards such as the *International Classification of Diseases (ICD)* enabling consistent reporting, research, and policy planning across institutions.[4]

In public health, accurate classification helps identify emerging threats, monitor morbidity and mortality, and evaluate the effectiveness of interventions. Using this dataset, the thesis explores how NLP models can improve automatic categorization of medical texts, which is crucial for scaling health surveillance and enhancing clinical documentation systems.

[4]See: `https://www.who.int/standards/classifications/classification-of-diseases`

Chapter 1

Theoretical and Technical Background

1.1 NLP in the Biomedical Domain

One area within (Artificial Intelligence (AI)) involves teaching machines to process and understand human language—commonly referred to as Natural Language Processing (NLP). While NLP methods have been widely adopted across various domains, their application in biomedical contexts remains particularly demanding. Texts in this domain, including clinical records, scientific publications, and research summaries, often feature highly technical vocabulary, sophisticated grammatical structures, and constantly changing terminology, all of which complicate language modeling for machines.

Biomedical documents merge medical, biological, and technical content. They frequently contain domain-specific expressions such as disease names, medications, therapeutic approaches, biological pathways, and clinical findings. These terms are often absent from general-purpose language corpora used to train many NLP models. Additionally, such texts are rich in acronyms, abbreviations, and rare terms that are not always clarified within the text. As a result, conventional NLP systems trained on general data sources like social media or news articles may underperform when applied to biomedical language [2].

In earlier stages of biomedical text processing, simpler yet effective representation techniques such as Term Frequency–Inverse Document Frequency (TF-IDF) played a foundational role. TF-IDF transforms textual data into numerical vectors by quantifying how important a word is within a document relative to a larger corpus. This approach helps highlight domain-relevant terms (e.g., specific gene or disease names) that might otherwise be overlooked in frequency-based models. Despite its simplicity, TF-IDF has proven useful for tasks like document classification, information retrieval, and keyword extraction in biomedical contexts.

When combined with traditional machine learning algorithms such as Support Vector Machines (SVM), Naive Bayes, or logistic regression, TF-IDF representations can deliver competitive performance for various supervised learning problems. These models offer interpretability and computational efficiency, making them suitable for situations where large-scale neural architectures are impractical or when domain experts require transparent decision rules. However, TF-IDF lacks the capacity to capture semantic relationships between terms, limiting its effectiveness in tasks requiring deeper contextual understanding—such as relation extraction or concept disambiguation.

The biomedical field generates an immense volume of unstructured textual data, including scientific articles, research abstracts, clinical notes, and database entries. To extract meaningful insights, this information needs to be organized, interpreted, and contextualized. Tasks such as literature mining, clinical decision support, or epidemiological analysis depend heavily on this process. However, unstructured texts introduce additional challenges like linguistic ambiguity and inconsistency. For instance, a single term may refer to entirely different concepts depending on the context (e.g., “cell” might mean a biological unit or a telecommunications structure).

Another key difficulty in biomedical NLP lies in the fast pace of medical innovation. New discoveries, medications, diagnostic methods, and treatment protocols are published frequently. Therefore, NLP systems must be able to incorporate and respond to emerging knowledge effectively. As terminology and conceptual frameworks evolve, models and embeddings must be continually updated to reflect these

shifts.

Given the complexity of biomedical language, traditional representation techniques such as vector-based models (Fasttext, GloVe) often fall short of capturing domain-specific nuances. To overcome these limitations, domain-specific embeddings such as BioMedNLP have been developed. These embeddings are pretrained on large-scale biomedical corpora, including PubMed and clinical datasets, allowing them to better capture the semantic relationships and terminology unique to the biomedical domain. In our work, we utilized BioMedNLP embeddings to represent abstract texts more effectively before feeding them into downstream models. More advanced architectures—including recurrent neural networks (Recurrent Neural Network (RNN), LSTM, GRU) and attention-based mechanisms—have become essential for modeling long-range dependencies and intricate language patterns.

Automating the processing of biomedical texts has far-reaching applications. In scientific research, it facilitates the identification and classification of relevant literature in specialized areas such as rare diseases or novel therapies. In clinical environments, it can assist in enhancing decision-making by extracting pertinent data from patient records and medical studies.

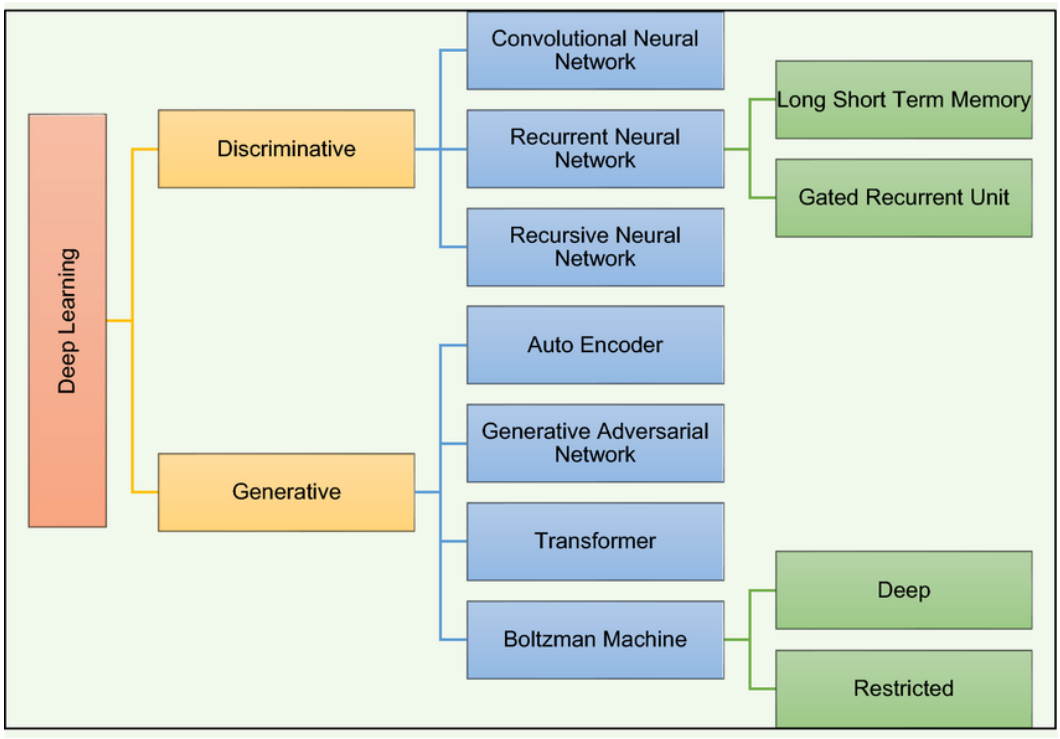


Figure 1.1: Classification of deep learning models used in natural language processing. Source: `https://sl.bing.net/gQNjaTwIKIe`

1.2 Machine Learning techniques

Before delving into deep learning approaches, it is important to briefly review some of the foundational machine learning techniques that have significantly influenced the development of biomedical NLP. These traditional models, though simpler, provide critical insights into how structured patterns can be learned from textual data. They often serve as effective baselines and remain valuable in scenarios with limited data, computational constraints, or when interpretability is essential. In this section, we explore key machine learning algorithms such as Support Vector Machines and Random Forests, particularly in the context of processing biomedical texts.

1.2.1 Linear Support Vector Machine

Support Vector Machines (SVMs) are supervised learning models that have proven to be particularly effective for text classification tasks due to their ability to handle high-dimensional feature spaces, such as

those generated by TF-IDF representations. An SVM aims to find the optimal hyperplane that separates data points of different classes with the maximum margin. In the context of textual data, this translates to distinguishing documents based on the presence or absence of informative terms.

Formally, the linear SVM for binary classification can be described by the following unconstrained optimization problem using the hinge loss:

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)) \tag{1.1}$$

Here, $\mathbf{x}_i \in \mathbb{R}^d$ are the input feature vectors (e.g., TF-IDF), $y_i \in \{-1, +1\}$ are the class labels, $\mathbf{w}$ is the weight vector, b is the bias term, and $C > 0$ is the regularization parameter that balances margin maximization and misclassification penalty.

While the above formulation is specific to binary classification, SVMs can be extended to multi-class problems using strategies such as one-vs-rest (OvR) or one-vs-one (OvO), which decompose the task into multiple binary classification subproblems.

In biomedical NLP, where documents often contain sparse yet highly discriminative terminology, SVMs are advantageous because they can focus on boundary-defining terms while ignoring irrelevant features. This makes them highly suitable for classification problems involving medical literature, clinical notes, or disease tagging.

1.2.2 Random Forest

Random Forest is another popular machine learning algorithm used for classification and regression tasks, including those involving biomedical texts. It is an ensemble method that builds multiple decision trees during training and outputs the class that is the mode of the classes (for classification) or mean prediction (for regression) of the individual trees.

Each decision tree in the forest is built by recursively splitting the training data based on features that maximize information gain. A common criterion for classification is the minimization of the Gini impurity, defined as:

$$G(t) = 1 - \sum_{k=1}^{K} p_k^2 \tag{1.2}$$

where $G(t)$ is the Gini impurity of node t, p_k is the proportion of samples belonging to class k at node t, and K is the total number of classes.

The overall goal is to construct splits that reduce impurity as much as possible across the trees in the ensemble. The final decision is made by aggregating the predictions from all individual trees, which helps reduce variance and improve generalization.

Random Forests are known for their robustness to overfitting, their ability to handle noisy data, and their natural capacity to model feature interactions without the need for extensive feature engineering. In biomedical NLP, they are commonly employed for tasks such as disease mention detection, gene classification, or document triage. When combined with vector representations such as TF-IDF, Random Forests can achieve competitive results while remaining relatively easy to implement and interpret.

1.3 Sequential Architectures (RNN, LSTM, GRU)

1.3.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) are a class of models designed for processing sequential data. Unlike feedforward networks, RNN incorporate a recurrent loop that allows information to be passed from one time step to the next, making them suitable for modeling temporal dependencies in data. They are distinguished by their ability to remember previous states in a sequence. At each time step t, the hidden state h_t is computed from the input x_t and the previous state h_{t-1}, according to the following equation:

$$h_t = f(Wx_t + Uh_{t-1} + b)$$

where W and U are weight matrices associated respectively with the input and the previous state, b is a bias term, and f is an activation function, usually a tanh or sigmoid function. This structure enables RNN to accumulate contextual memory over the sequence.

Thanks to this architecture, RNN are capable of capturing temporal relationships within a sequence. This makes them particularly suited for processing ordered data streams such as time series, text, or audio signals. Their effectiveness relies on their ability to handle variable-length sequences while maintaining an internal state that summarizes past information.

However, despite their ability to handle sequential data, classical RNN have major limitations. They suffer notably from the vanishing and exploding gradient problems, which can occur when training on long sequences. The former makes it difficult to adjust weights in deep layers, while the latter leads to unstable updates of model parameters. These issues reduce the effectiveness of RNN in modeling long-term dependencies and have led to the development of improved variants such as LSTM and GRU.

To better visualize the working principle of RNN, the following figure illustrates the recurrent connections between the input x_t, hidden state h_t, output L_t, and information flow across time steps.

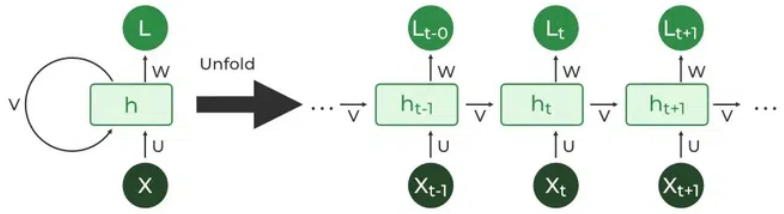


Figure 1.2: Illustration of the operation of Recurrent Neural Networks (RNN). Source: `https://miro.medium.com/v2/resize:fit:660/1*uLTBA8Myf6_IwtpfLr4Xpg.png`

1.3.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory networks (LSTM) are an advanced type of Recurrent Neural Network (RNN) specifically developed to address the challenges associated with learning long-range dependencies, such as vanishing and exploding gradients. This architecture incorporates a memory cell that enables the network to retain relevant information across extended sequences, with information flow regulated through a set of gating mechanisms.

At each time step t, the fundamental operations of an LSTM cell are described by the following equations:

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) && \text{(Forget gate)} \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) && \text{(Input gate)} \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) && \text{(Candidate cell state)} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{(Cell state update)} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) && \text{(Output gate)} \\
h_t &= o_t \odot \tanh(c_t) && \text{(Hidden state)}
\end{aligned}
$$

where:

- x_t is the input at time t;
- h_{t-1} is the previous hidden state;
- f_t, i_t, and o_t are the forget, input, and output gate vectors respectively;
- c_t is the memory cell state at time t;

- $\tilde{c}_t$ is the proposed candidate cell state;
- σ is the sigmoid function, tanh the hyperbolic tangent, and $\odot$ denotes element-wise multiplication.

This structure allows LSTM to dynamically decide which information to keep, update, or output, offering selective memory capacity very useful for processing complex sequences. Thanks to this flexibility, LSTM have become a standard in many tasks such as language modeling, machine translation, and time series analysis.

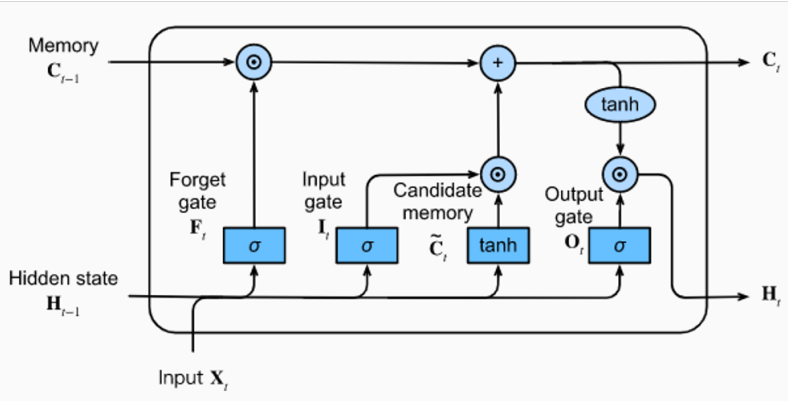


Figure 1.3: Logic of the LSTM architecture. Source: `https://sl.bing.net/bWWTQ1HkC7M`

Although powerful, LSTM have considerable computational complexity due to the large number of parameters associated with their multiple gates. To provide a lighter yet effective alternative, the community introduced a simplified architecture: the Gated Recurrent Unit (GRU).

1.3.3 Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) is a streamlined alternative to traditional RNNs, designed to offer similar performance to LSTMs with a less complex architecture. Instead of maintaining separate mechanisms for forgetting and updating information, GRUs combine these roles into a single update gate. Additionally, they eliminate the separate memory cell used in LSTMs, relying solely on the hidden state to carry information. This design reduces computational overhead and can lead to faster training in various applications.

The equations governing a GRU cell are as follows:

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) && \text{(Update gate)} \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) && \text{(Reset gate)} \\
\tilde{h}_t &= \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) && \text{(Candidate state)} \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t && \text{(Hidden state)}
\end{aligned}
$$

where:

- z_t is the update gate;
- r_t is the reset gate;
- $\tilde{h}_t$ is the candidate hidden state;
- h_t is the final hidden state of the GRU cell at time t.

This compact architecture allows GRU to effectively adapt to temporal dependencies while limiting computational load.

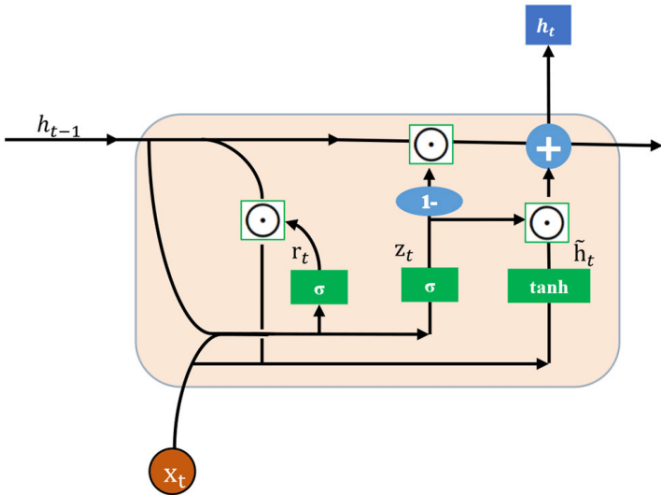


Figure 1.4: Illustration of the Gated Recurrent Unit (GRU). Source: `https://sl.bing.net/bWWTQ1HkC7M`

This compact architecture allows GRU to effectively adapt to temporal dependencies while limiting computational load.

However, despite their ability to capture long-term relationships, sequential architectures such as LSTM and GRU process sequences uniformly, without giving more importance to certain parts of the text over others. To overcome this limitation, attention mechanisms have been introduced to enable the model to dynamically focus on the most relevant elements of the sequence.

1.3.4 Bahdanau Attention Mechanism

The Bahdanau attention mechanism, introduced in the context of neural machine translation [1] (figure 1.5), aims to overcome the limitations of classical sequential models that compress all the information of a sequence into a fixed-length vector. Such compression can be suboptimal, especially when sequences are long or contain dispersed information.

Attention allows the model to dynamically weight the elements of the sequence according to their relevance to the task at hand. In other words, instead of treating all parts of a text equally, the model learns to focus more on the parts that are relevant for prediction.

In this thesis, a simplified version of Bahdanau's mechanism is applied on the outputs of a bidirectional GRU. Specifically, each time-step output of the GRU is transformed via a feedforward network to produce an importance score (or attention score). These scores are then normalized through a softmax function to obtain attention weights. The context vector is then computed as a weighted average of the hidden states of the sequence, where the weights reflect the relevance of each word.

The hidden vector at time t in a bidirectional GRU is defined as the concatenation of the forward and backward vectors:

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

where $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ are respectively the hidden states generated by the forward and backward passes of the GRU.

The attention score for each hidden state at time t is computed as follows:

$$\text{score}_t = \mathbf{v}^\top \tanh(\mathbf{W}[\overrightarrow{h_t}; \overleftarrow{h_t}])$$

The relative importance of each element in the sequence is then calculated by a softmax function:

$$\alpha_t = \frac{\exp(\text{score}_t)}{\sum_{t'} \exp(\text{score}_{t'})}$$

Finally, the context vector, summarizing the relevant information from the hidden states of the sequence, is obtained by a weighted sum of the hidden states:

$$\text{context} = \sum_t \alpha_t [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

In these equations:

- h_t is the hidden vector at time t, which is the concatenation of the forward and backward hidden states;
- α_t is the attention weight associated with the hidden state h_t;
- context is the resulting vector, which summarizes the sequence by emphasizing the most relevant elements.

This mechanism enables the model to leverage both preceding and succeeding contextual information in a sequence, which is particularly useful for tasks such as biomedical text classification, where a term may have different meanings depending on the surrounding context.

After exploring Bahdanau's attention mechanism and its application in sequential models such as the bidirectional GRU, it is important to turn to another essential component in natural language processing architectures: the vector representation of words and sentences. These representations play a fundamental role in capturing semantic and contextual relationships between terms.

In this regard, several vector representation techniques have emerged to better understand and model language. Among the most popular are GloVe and Fasttext. These approaches have helped to overcome the limitations of classical representations by integrating both global information and contextual information.

The next section presents these various vector representation techniques, with a particular focus on their application in the biomedical field, where terms are often highly specialized and context-dependent.

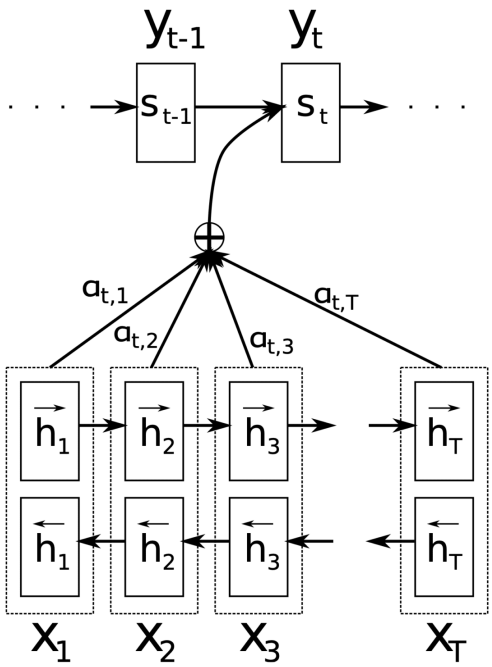


Figure 1.5: Bahdanau attention mechanism illustration. Source: `https://sl.bing.net/eAB8rGLRvg`

1.4 Vector Representations

The effectiveness of natural language processing (NLP) models is closely tied to the way textual data is transformed into numerical form. Since most machine learning algorithms—including Support Vector Machines (SVM) and Random Forests—require fixed-size numerical input, raw text must be converted into vector representations before training can occur.

A range of techniques exists for this transformation. One of the earliest and most commonly used families of methods is based on term-level statistics, such as *Term Frequency* (TF) and *Inverse Document Frequency* (IDF). These approaches emphasize the importance of words that are frequent in a document but rare across the corpus, allowing models to focus on discriminative terms.

Beyond statistical representations, *static word embeddings* have also played a central role in NLP. These assign each word a fixed-dimensional vector that captures semantic similarity based on co-occurrence patterns. In this section, we first present classical representations like TF-IDF, followed by a discussion of prominent embedding techniques such as GloVe and Fasttext, which significantly improved the modeling of semantic relationships.

1.4.1 Term Frequency-Inverse Document Frequency

TF-IDF (Term Frequency–Inverse Document Frequency) is a widely used vectorization technique that transforms text into numerical feature vectors based on word importance. The term frequency component reflects how often a word appears in a document, while the inverse document frequency down-weights terms that are common across many documents, reducing their influence on the final representation.

Mathematically, for a term t in document d, the TF-IDF score is computed as:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \cdot \log\left(\frac{N}{\text{df}(t)}\right)$$

where $\text{tf}(t, d)$ is the frequency of term t in document d, N is the total number of documents, and $\text{df}(t)$ is the number of documents in which t appears. This representation results in sparse, high-dimensional vectors, which are particularly effective for traditional machine learning models like SVMs and Random Forests due to their ability to handle such input spaces efficiently.

1.4.2 Word Embeddings: GloVe and FastText

Static *word embeddings* represent each word as a dense vector in a continuous low-dimensional space. These representations are learned from large text corpora by exploiting word co-occurrences. Two of the most influential approaches in this area are GloVe and Fasttext.

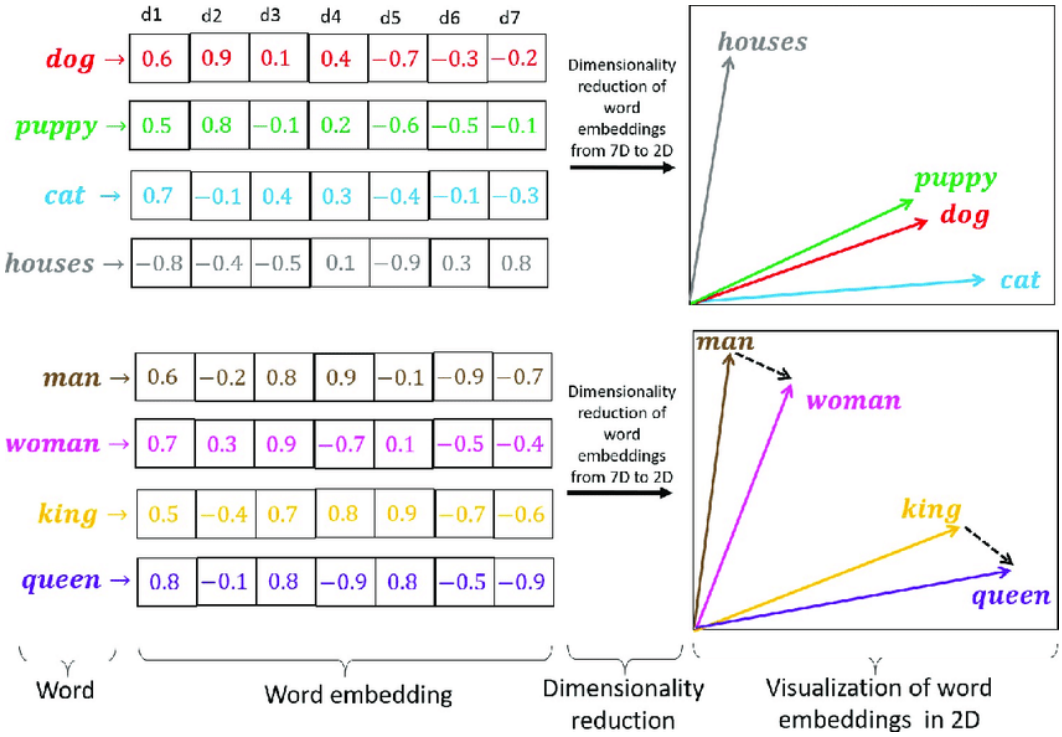


Figure 1.6: Illustration of the principle of *word embeddings*: similar words are projected into nearby regions of the vector space. Source: `https://sl.bing.net/jumSq5RbE1Q`

1.4.2.1 Fasttext

FastText is an enhanced version of the Word2Vec model that incorporates subword information, allowing it to generate more accurate vector representations—especially for rare or morphologically complex words. Unlike GloVe or Method for learning word vector representations developed by Mikolov et al. (Word2Vec), which treat each word as an atomic unit, Fasttext represents a word as the sum of vectors of its subunits called character *n-grams*. Formally, if a word w consists of n-grams $g_1, g_2, \ldots, g_K$, then its vector representation $\mathbf{v}_w$ is defined as:

$$\mathbf{v}_w = \sum_{k=1}^{K} \mathbf{z}_{g_k}$$

where $\mathbf{z}_{g_k}$ is the vector associated with the n-gram g_k.

For example, the word "cat" with trigrams (n=3) would be represented by the following n-grams: _ca, cat, at_. Fasttext also adds word boundary markers ("_" at the beginning and end) to distinguish positions within the word.

This representation allows the model to share information among words with similar morphology, making it robust to rare or out-of-vocabulary words. Additionally, FastText can produce vectors for words missing from the training vocabulary as long as their subunits were observed.

By preserving the semantic properties of Word2Vec while capturing morphological regularities (prefixes, suffixes, roots), FastText is especially well-suited for specialized domains like biomedicine, where terms are often long, compound, and rare.

1.4.2.2 GloVe

GloVe [9] is an unsupervised learning method that combines the advantages of local context-window based approaches (like Word2Vec) with global statistical methods. Unlike Word2Vec, which optimizes a local objective function, GloVe takes into account global information about word distribution across the entire corpus. It relies on constructing a word co-occurrence matrix, where each entry X_{ij} corresponds to the number of times word j appears in the context of word i, i.e., a measure of word co-occurrence frequency.

The goal of the model is to learn vectors w_i and $\tilde{w}_j$ for each word i and context j such that their dot product approximates the logarithm of X_{ij}. This relationship can be expressed as:

$$w_i^\top \tilde{w}_j + b_i + \tilde{b}_j \approx \log(X_{ij})$$

where b_i and $\tilde{b}_j$ are learned biases for each word and context respectively. The model is trained to optimize this relationship over all frequent word pairs in the corpus. Optimizing this objective captures semantic and syntactic relationships between words. In particular, vectors learned by GloVe can capture analogies such as:

$$\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}} \approx \vec{\text{queen}}$$

The embeddings produced by GloVe thus represent complex relationships between words, not only semantic but also analogical, making them especially useful in NLP applications such as text classification, information retrieval, and text generation.

A key advantage of GloVe is that it combines local co-occurrence information with a global corpus representation, enabling learning word vectors that capture both close semantic relations and more abstract relationships like analogies. Unlike local models focused solely on word relations within a fixed window, GloVe integrates the entire global co-occurrence structure.

However, a limitation of GloVe lies in its inability to handle contextual ambiguities of words. For example, the word "bank" can have very different meanings depending on context, but GloVe produces a single vector representation that does not reflect these variations. This lack of contextualization has motivated the emergence of more advanced models, such as contextual word representation models (e.g., Bidirectional Encoder Representations from Transformers (BERT)), which address this problem by generating different embeddings depending on the context in which the word appears.

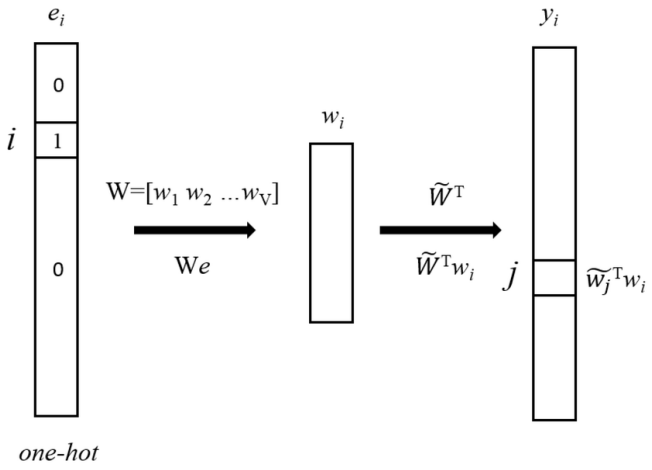


Figure 1.7: Architecture of the GloVe model, illustrating vector construction from co-occurrences. The input is a one-hot representation of a word at a given time. Word embedding matrices are used as weight matrices, and the model output is a vector obtained by the dot product between word vectors. Source: `https://www.researchgate.net/figure/The-model-architecture-of-GloVe-The-input-is-a-one-hot-representation-of-a-word-The_fig1_337461648`

1.4.3 Contextual Embeddings: BioMedNLP

Traditional word embeddings like GloVe and FastText assign fixed vectors to words, meaning a word always has the same representation, regardless of its context. This can be problematic in biomedical texts, where many terms have multiple meanings depending on how and where they are used. To overcome this, more recent models generate word embeddings that change based on surrounding words, enabling them to better capture meaning in context.

Contextual embeddings rely on deep learning models known as transformers, which process entire sequences of text rather than isolated words. In the biomedical domain, specialized transformer-based models have been developed and trained on large collections of scientific literature and clinical documents. These models—part of what we refer to here as the BioMedNLP framework—are able to understand technical vocabulary, abbreviations, and complex phrasing that are common in medical and biological writing.

In this work, we use contextual embeddings from such models to represent the text. These embeddings are especially useful for handling specialized or ambiguous terms. For example, a word like “blocker” may carry different meanings depending on whether the context is cardiovascular health or molecular biology. BioMedNLP embeddings help disambiguate such terms by adjusting their representation based on usage.

1.5 Visualization of Data Representations with t-SNE

Visualizing vector representations is an important step to better understand how a natural language processing model learns to distinguish different classes from textual data. In this project, we use the t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm to project high-dimensional vectors into a two-dimensional space for intuitive visualization.

t-SNE (t-distributed Stochastic Neighbor Embedding) is based on a probabilistic modeling of proximity between points. In the original high-dimensional space, the similarity between two points x_i and x_j is defined as a conditional probability $p_{j|i}$, given by:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)} \tag{1.3}$$

The joint symmetric probability is then defined as:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{1.4}$$

where n is the total number of points. In the low-dimensional projected space, a distribution q_{ij} is defined on the projected points y_i and y_j using a Student's t-distribution with one degree of freedom (which has heavier tails than a Gaussian):

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}} \tag{1.5}$$

The objective of t-SNE is then to minimize the Kullback-Leibler (KL) divergence between these two similarity distributions:

$$\mathcal{L} = KL(P\|Q) = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right) \tag{1.6}$$

This minimization is performed by gradient descent, and allows preservation of the local structures of the data — pairs of points that are close in the high-dimensional space remain close in the projected 2D space.

Applied at different stages of the pipeline, t-SNE provides a visual representation of the data as perceived by the model — whether at the input (for example, GloVe or BioBERT, a BERT variant for the biomedical domain (BioBERT) embeddings) or through intermediate activations after passing through model layers (LSTM, attention, etc.).

The interpretation of t-SNE projections relies on the preservation of local data structure. In other words, if two points are close in the projected space, they were likely also close in the original high-dimensional space. Thus, a clear visual separation between groups (or classes) in the t-SNE map may indicate that the model has learned discriminative representations. Conversely, significant overlap may signal that the classes are not well separable in the learned space, which can result in classification errors.

In this work, t-SNE allowed us to visualize clusters of data from different classes, identify potential zones of confusion between classes, and observe the evolution of the representations learned by the model throughout training.

This visualization is not merely illustrative: it offers a qualitative insight into the model's behavior and can guide methodological choices, for example by suggesting the introduction of rebalancing techniques or adjustments to the architecture if classes remain difficult to separate.

1.6 Few-Shot Learning in NLP: Definitions, Methods, and Challenges

Few-shot learning (Few-Shot Learning (FSL)) represents an approach where models must learn effectively from only a few examples per class, a scenario commonly encountered in contexts where obtaining a sufficient amount of annotated data is difficult. This approach is particularly useful in NLP, where datasets can be scarce or hard to manually annotate, such as in domain-specific applications like biomedicine or law.

1.6.1 Definitions and Concepts

Few-shot learning is based on the idea that a model can learn complex tasks using very few examples. This capability is essential to overcome the limitations of traditional methods that require large amounts of annotated data to achieve good performance. In this context, a k-shot refers to the number of examples used per class to train a model. For example, in a 5-shot learning problem, the model is trained with five examples per class. A key concept in FSL is the ability of models to generalize from few examples.

1.6.2 Challenges in Few-Shot Learning

One major challenge in few-shot learning is generalization. The model must be able to make reliable predictions despite the limited number of training examples. This difficulty is heightened in specific

domains such as biomedicine, where semantic richness and term polysemy make the model prone to errors if the context is not well understood [11]. Techniques like multitask training and transfer learning can help overcome these obstacles by leveraging information from related tasks or domains.

Class contrast in a Few-Shot dataset can also be challenging. Models need to distinguish very similar examples while being robust to variations in input data. Approaches such as data augmentation and regularization techniques can be useful to improve model robustness in FSL.

1.7 Class Imbalance Issues and Adaptation Techniques

Class imbalance is a common problem in biomedical text classification, where some classes are overrepresented while others, often rarer but clinically important, are underrepresented. This disparity can heavily bias model learning, which tends to favor the majority class at the expense of accuracy on minority classes. To address this, several adaptation techniques have been developed, including resampling methods and class weighting strategies in the loss function. These techniques are particularly relevant in low-data or highly imbalanced contexts, such as in biomedicine.

1.7.1 Resampling Methods

Resampling methods aim to modify the data distribution to compensate for class imbalance. There are mainly two approaches:

- **Oversampling**, which artificially increases the proportion of minority class examples;
- **Undersampling**, which reduces the size of majority classes to balance the distribution.

Although both types of methods are commonly used in the literature, our project focuses exclusively on **oversampling** techniques. This choice is based on a review of specialized articles on learning from imbalanced data, particularly in the biomedical domain, where preserving majority class data is often crucial to maintain the richness of clinical cases. Moreover, oversampling methods enrich the minority class without altering the overall structure of the dataset.

Among oversampling approaches, we selected widely recognized technique: SMOTE (Synthetic Minority Over-sampling Technique) [3]. This method has demonstrated its effectiveness in several studies to improve model performance on highly imbalanced datasets.

1.7.1.1 SMOTE — Synthetic Minority Over-sampling Technique

SMOTE [3] is an oversampling method designed to address class imbalance issues in datasets, particularly where minority classes are underrepresented. Unlike simpler approaches such as random duplication of minority examples, SMOTE generates new synthetic examples by interpolation, introducing greater diversity in the data.

The fundamental principle of SMOTE consists in randomly selecting one of the k nearest neighbors of a minority sample, then creating a new example along the line segment connecting these two points in feature space. Formally, if x_i is a minority example and $x_i^{(NN)}$ one of its nearest neighbors, a new synthetic example x_{new} is generated as:

$$x_{\text{new}} = x_i + \delta \cdot (x_i^{(NN)} - x_i), \quad \text{where } \delta \sim \mathcal{U}(0,1) \tag{1.7}$$

where δ is a random weighting factor drawn from a uniform distribution. This linear interpolation produces instances close to existing data, enriching the minority class feature space more smoothly than exact duplications.

This technique has several advantages: it reduces the risk of overfitting, improves decision boundary definition, and allows classifiers to better generalize in low-density areas. It is particularly relevant in biomedical contexts where rare conditions or specific diseases are underrepresented.

However, SMOTE also has limitations. It may create non-representative synthetic instances if neighbors used are near class boundaries, which can introduce noise. Additionally, by artificially extending the minority class, it can exacerbate class overlap if the original problem is not linearly separable. Variants

such as Borderline-SMOTE have been proposed to address these issues by focusing synthetic example generation on the most ambiguous regions.

The principle of SMOTE is illustrated in Figure 1.8, which presents the algorithm as pseudocode. It details the steps to generate synthetic examples via random interpolation between a minority instance and its nearest neighbors. This method can be combined with other techniques, such as majority class undersampling or loss function weighting, to optimize performance on imbalanced datasets.

Algorithm 1 SMOTE algorithm

```
1: function SMOTE(T, N, k)
   Input: T; N; k        ▷ #minority class examples, Amount of oversampling, #nearest
   neighbors
   Output: (N/100) * T synthetic minority class samples
   Variables: Sample[][]: array for original minority class samples;
   newindex: keeps a count of number of synthetic samples generated, initialized to 0;
   Synthetic[][]: array for synthetic samples
2:     if N < 100 then
3:         Randomize the T minority class samples
4:         T = (N/100)*T
5:         N = 100
6:     end if
7:     N = (int)N/100    ▷ The amount of SMOTE is assumed to be in integral multiples
   of 100.
8:     for i = 1 to T do
9:         Compute k nearest neighbors for i, and save the indices in the nnarray
10:        POPULATE(N, i, nnarray)
11:    end for
12: end function
```

Figure 1.8: SMOTE algorithm: generation of synthetic examples from nearest neighbors of a minority instance. Adapted from [3].

1.7.2 Class Weighting in the Loss Function

In many real-world classification tasks, certain classes may appear far more frequently than others. This imbalance can lead machine learning models to favor the majority class, resulting in poor performance on the underrepresented categories. Class weighting is a technique designed to mitigate this issue by adjusting the influence each class has during training.

Instead of altering the data distribution through resampling, this method modifies the loss function by assigning greater importance to less frequent classes. By doing so, the model becomes more sensitive to errors involving minority classes, helping it learn to distinguish them more effectively.

This strategy is particularly useful in domains such as healthcare, fraud detection, or social sciences, where some classes—despite being rare—are critical and must be identified accurately. For instance, failing to detect a rare disease in a clinical setting could have far more serious consequences than misclassifying a common condition.

In multi-class classification problems, the idea extends naturally by assigning a specific weight to each class based on its frequency or significance. These weights guide the learning process, encouraging the model to allocate more attention to classes that are otherwise overlooked.

More advanced techniques also exist to refine this process further. Some loss functions introduce additional mechanisms to focus learning on challenging or ambiguous examples, offering even greater robustness in highly imbalanced datasets.

Overall, class weighting provides an effective and interpretable way to address class imbalance during model training, making it a practical choice in many high-stakes applications.

Chapter 2

Related Work

2.1 Overview of Biomedical Text Classification Methods

Automatic classification of biomedical texts is a central field in natural language processing (NLP), particularly crucial for applications such as the analysis of scientific publications, clinical information extraction, and decision support systems. This field has evolved alongside advances in machine learning, especially deep learning. The objective of this section is to present the evolution of biomedical text classification approaches, focusing on the major works that have influenced this domain, while highlighting the motivations that led to this thesis work.

In the early years of automatic text classification development, models were mainly based on simple representations such as the *bag-of-words*. Traditional approaches also included methods like support vector machines (SVM) and Naïve Bayes classifiers. Although these techniques were robust, they struggled to capture the complex contextual relationships between terms, which are essential in the biomedical domain. Biomedical corpora, often characterized by highly specialized vocabulary and significant semantic ambiguity, required a more sophisticated approach capable of understanding these subtle relations.The rise of sequential models in natural language processing gained momentum during the 2010s with the introduction of recurrent neural networks (RNN) and their more advanced variants such as LSTM and GRU. These architectures made it possible to model the sequential and contextual nature of language more effectively, as they take into account the order of words and the surrounding context within a sentence or document. This capability is particularly important in fields like biomedicine, where the meaning of a term can vary significantly depending on its context.

Basic RNNs were the first to address sequential data by using feedback connections to retain information across time steps. However, they faced challenges when processing long sequences due to difficulties in maintaining gradients during training. To address this, LSTMs were developed with internal memory units that better preserve long-term information, making them more suitable for capturing distant word relationships.

Later, GRUs were introduced as a simplified alternative to LSTMs. While maintaining a similar level of performance, GRUs reduce architectural complexity and computational cost by using fewer parameters, making them attractive for various practical applications.

In parallel with these advances, static word embedding techniques such as Fasttext and GloVe emerged, offering a more semantically meaningful way to represent words. These dense vector representations replaced earlier frequency-based approaches by capturing word similarities based on their usage patterns in large text corpora. In specialized domains like healthcare, domain-specific embeddings further improved representation quality by incorporating vocabulary and linguistic structures unique to medical texts.

2.1.1 Support Vector Machine Active Learning with Applications to Text Classification

Tong and Koller [4]introduced an innovative active learning method specifically tailored for Support Vector Machines (SVMs). By exploiting the relationship between the parameter space and the feature space, they formulated three variants of algorithms aimed at minimizing the version space at each annotation query. Their empirical evaluation on the Reuters-21578 and Newsgroups datasets revealed that these

approaches achieve performance comparable to that obtained with a fully annotated dataset, while significantly reducing the need for labeled data—sometimes by an order of magnitude. They also show that having a larger set of unlabeled data improves the quality of the final classifier. Among the proposed methods, the so-called Simple strategy stands out for its speed, although it may be unstable on certain topics. The MaxMin and Ratio variants, while more computationally expensive, offer better robustness when annotation cost is high. Finally, the authors suggest several avenues for improvement, including the integration of prior knowledge and the use of Bayesian frameworks to guide instance selection more optimally.

To evaluate their methods, the authors conducted experiments on the top ten most frequently occurring topics. For each topic, they sampled a pool of 1000 unlabeled documents, from which two were randomly selected and labeled—one relevant to the topic and one not. This minimal labeled set served as the initial training data, while the remaining 998 documents were used as the unlabeled pool. After a fixed number of queries, each learner (a SVM with a polynomial kernel of degree one) was trained and evaluated on an independent test set. This process was repeated thirty times per topic, and the results were averaged to ensure statistical robustness.

Topic	Simple	MaxMin	Ratio	Equivalent Random size
Earn	86.39 ± 1.65	87.75 ± 1.40	90.24 ± 2.31	34
Acq	77.04 ± 1.17	77.08 ± 2.00	80.42 ± 1.50	> 100
Money-fx	93.82 ± 0.35	$\mathbf{94.80 \pm 0.14}$	$\mathbf{94.83 \pm 0.13}$	50
Grain	95.53 ± 0.09	95.29 ± 0.38	95.55 ± 1.22	13
Crude	95.26 ± 0.38	95.26 ± 0.15	95.35 ± 0.21	> 100
Trade	96.31 ± 0.28	96.64 ± 0.10	96.60 ± 0.15	> 100
Interest	96.15 ± 0.21	96.55 ± 0.09	96.43 ± 0.09	> 100
Ship	97.75 ± 0.11	97.81 ± 0.09	97.66 ± 0.12	> 100
Wheat	98.10 ± 0.24	98.48 ± 0.09	98.13 ± 0.20	> 100
Corn	98.31 ± 0.19	98.56 ± 0.05	98.30 ± 0.19	15

Figure 2.1: Average test set accuracy over the top ten most frequently occurring topics (most frequent topic first) when trained with ten labeled documents. Boldface indicates statistical significance.

Figure 2.1 presents the average test set accuracy for the different querying strategies: Simple Margin, MaxMin Margin, Ratio Margin, and Random Sampling. The Random Sample method simulates a passive learning scenario, where the training set is built from randomly selected labeled data. The active learning methods—particularly MaxMin and Ratio—consistently outperform random sampling, confirming the effectiveness of selecting informative instances. Notably, even with only ten labeled documents, the classifiers learned through active selection achieve significantly better accuracy, underlining the potential of active learning in data-scarce environments.

2.1.2 A Survey of Text Classification Algorithms

Kowsari et al. [7]provide a comprehensive survey of text classification algorithms, highlighting the standard pipeline involved in automated text categorization. The pipeline consists of four main stages: feature extraction, dimensionality reduction, classification, and evaluation. The paper reviews traditional approaches such as Naïve Bayes, Support Vector Machines (SVM), and k-Nearest Neighbors (k-NN), as well as ensemble methods like bagging and boosting, and recent advances in deep learning including CNNs, RNNs, and various word embedding techniques (e.g., Word2Vec, GloVe, Fasttext).

Their comparative analysis (table 2.2) emphasizes that the optimal choice of algorithm largely depends on the application domain, dataset size, and text characteristics. Deep learning models tend to outperform traditional techniques on large-scale corpora but require substantial labeled data. Furthermore, the quality of text representation is shown to be critical for performance across all methods.

The authors also discuss limitations such as imbalanced datasets, interpretability challenges, and poor generalization on short or ambiguous texts. They highlight real-world applications in areas like information retrieval, spam filtering, public health monitoring, and document management. Ultimately, they advocate for hybrid models that leverage the strengths of multiple classification techniques for

enhanced robustness and accuracy.

Model	Author(s)	Architecture	Novelty	Feature Extraction	Details	Corpus	Validation Measure	Limitation
Rocchio Algorithm	B.J. Sowmya et al. [106]	Hierarchical Rocchio	Classificationon hierarchical data	TF-IDF	Use CUDA on GPU to compute and compare the distances.	Wikipedia	F1-Macro	Works only on hierarchical data sets and retrieves a few relevant documents
Boosting	S. Bloehdorn et al. [114]		AdaBoost for with semantic features	BOW	Ensemble learning algorithm	Reuters-21578	F1-Macro and F1-Micro	Computational complexity and loss of interpretability
Logistic Regression	A. Genkin et al. [120]	Bayesian Logistic Regression	Logistic regression analysis of high-dimensional data	TF-IDF	It is based on Gaussian Priors and Ridge Logistic Regression	RCV1-v2	F1-Macro	Prediction outcomes is based on a set of independent variables
Naïve Bayes	Kim, S.B et al. [131]	Weight Enhancing Method	Multivariate poisson model for text Classification	Weights words	Per-document term frequency normalization to estimate the Poisson parameter	Reuters-21578	F1-Macro	This method makes a strong assumption about the shape of the data distribution
SVM and KNN	K. Chen et al. [148]	Inverse Gravity Moment	Introduced TFIGM (term frequency & inverse gravity moment)	TF-IDF and TFIGM	Incorporates a statistical model to precisely measure the class distinguishing power of a term	20 Newsgroups and Reuters-21578	F1-Macro	Fails to capture polysemy and also still semantic and sentatics is not solved
Support Vector Machines	H. Lodhi et al. [151]	String Subsequence Kernel	Use of a special kernel	Similarity using TF-IDF	The kernel is an inner product in the feature space generated by all subsequences of length k	Reuters-21578	F1-Macro	The lack of transparency in the results
Conditional Random Field (CRF)	T. Chen et al. [175]	BiLSTM-CRF	Apply a neural network based sequence model to classify opinionated sentences into three types according to the number of targets appearing in a sentence	Word embedding	Improve sentence-level sentiment analysis via sentence type classification	Customer reviews	Accuracy	High computational complexity and this algorithm does not perform with unseen words

Model	Author(s)	Architecture	Novelty	Feature Extraction	Details	Corpus	Validation Measure	Limitation
Deep Learning	Z. Yang et al. [193]	Hierarchical Attention Networks	It has a hierarchical structure	Word embedding	Two levels of attention mechanisms applied at the word and sentence-level	Yelp, IMDB review, and Amazon review	Accuracy	Works only for document-level
Deep Learning	J. Chen et al. [228]	Deep Neural Networks	Convolutional neural networks (CNN) using 2-dimensional TF-IDF features	2D TF-IDF	A new solution to the verbal aggression detection task	Twitter comments	F1-Macro and F1-Micro	Data dependent for designed a model architecture
Deep Learning	M. Jiang et al. [1]	Deep Belief Network	Hybrid text classification model based on deep belief network and softmax regression.	DBN	DBN completes the feature learning to solve the high dimension and sparse matrix problem and softmax regression is employed to classify the texts	Reuters-21578 and 20-Newsgroup	Error-rate	Computationally is expensive and model interpretability is still a problem of this model
Deep Learning	X. Zhang et al. [229]	CNN	Character-level convolutional networks (ConvNets) for text classification	Encoded Characters	Character-level ConvNet contains 6 convolutional layers and 3 fully-connected layers	Yelp, Amazon review and Yahoo! Answers data set	Relative errors	This model is only designed to discover position-invariant features of their inputs
Deep Learning	K. Kowsari [4]	Ensemble deep learning algorithm (CNN, DNN and RNN)	Solves the problem of finding the best deep learning structure and architecture	TF-IDF and GloVe	Random Multimodel Deep Learning (RDML)	IMDB review, Reuters-21578, 20NewsGroup, and WOS	Accuracy	Computationally is expensive
Deep Learning	K. Kowsari [2]	Hierarchical structure	Employs stacks of deep learning architectures to provide specialized understanding at each level of the document hierarchy	TF-IDF and GloVe	Hierarchical Deep Learning for Text Classification (HDLTex)	Web of science data set	Accuracy	Works only for hierarchical data sets

Figure 2.2: Comparison of the text classifcation techniques.

2.1.3 The Impact of Preprocessing on Text Classification

Uysal and Gunal [8]investigate how various text preprocessing techniques influence the performance of machine learning models in text classification tasks. Their study systematically evaluates preprocessing steps such as stop-word removal, stemming, lemmatization, lowercasing, and punctuation removal across several classifiers including Naïve Bayes, k-NN, SVM, and decision trees.

The authors conduct experiments on standard datasets such as Reuters-21578 and 20 Newsgroups, applying different combinations of preprocessing techniques. Results show that while some steps (like stop-word removal and lowercasing) consistently improve performance, others (like stemming and lemmatization) produce mixed results depending on the dataset and classifier. Furthermore, they observe that excessive preprocessing can degrade classification accuracy by removing semantically relevant features.

As illustrated in Figure 2.3, empirical evaluation on an English email dataset confirms that the impact of preprocessing varies notably across methods and configurations. The results highlight that preprocessing is not universally beneficial and should be customized to the specific characteristics of both

the dataset and the learning algorithm. The authors emphasize the importance of empirical testing when designing preprocessing pipelines for text classification.

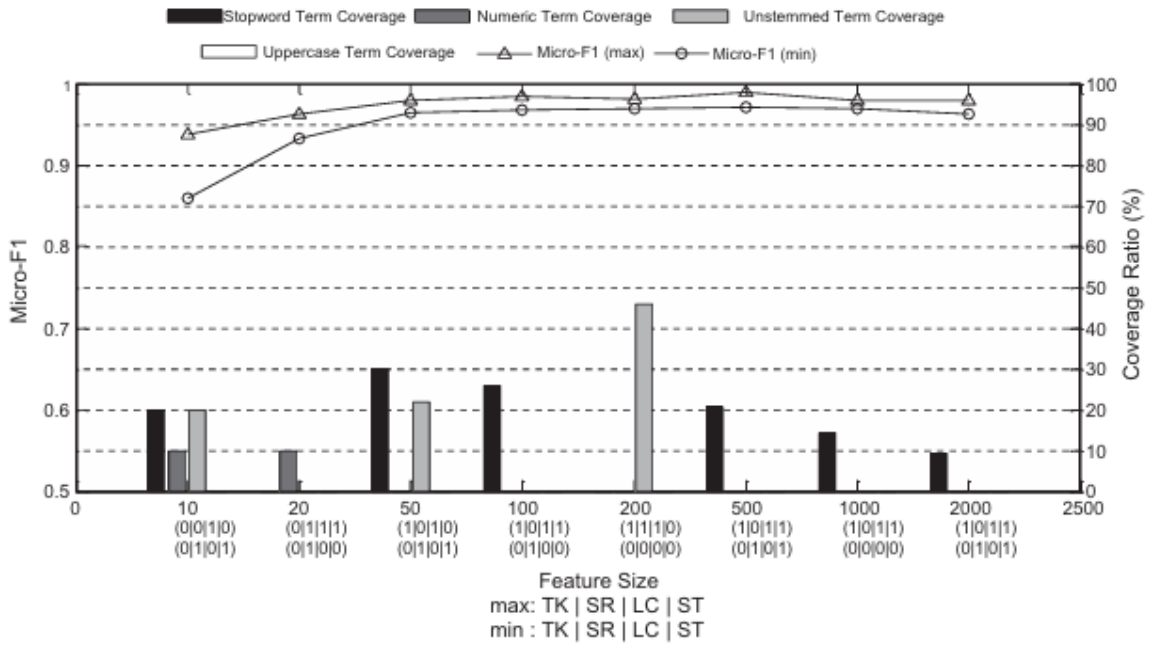


Figure 2.3: Experimental results for the English email dataset.

2.1.4 Conclusion

The studies presented in this chapter highlight the significant evolution of text classification approaches, particularly in the biomedical domain where the complexity of language and semantic richness demand advanced methods. From traditional models based on simple representations such as the bag-of-words to modern deep neural networks, progress in the field has substantially improved the ability of algorithms to capture intricate contextual relationships.

Research on active learning, such as that by Tong and Koller, demonstrates the effectiveness of selective annotation strategies in reducing the need for labeled data while maintaining high performance. Kowsari et al. offer a comprehensive overview of classification algorithms, emphasizing that the optimal choice of technique depends on application context, dataset size, and quality of textual representation. Meanwhile, the work of Uysal and Gunal reveals that text preprocessing—often treated as a standard step—has a substantial and sometimes variable impact on classification outcomes, underscoring the importance of empirical validation.

Collectively, these contributions point to the need for an adaptive framework that combines thoughtful model selection, effective linguistic representation, and tailored preprocessing strategies. It is within this context that our thesis is situated, aiming to leverage these advances to enhance biomedical text classification in real-world settings.

Chapter 3

Exploratory Data Analysis

3.1 Data Description

In this work, two distinct biomedical text classification datasets were employed.

The first dataset, provided by the **Memorial Sloan Kettering Cancer Center (MSKCC)** [5] [1], is an expert-annotated knowledge base in which world-class oncologists and researchers have manually labeled thousands of genetic mutations. This resource offers high-quality, domain-specific supervision that closely reflects real clinical decision-making, and it contains a large volume of tokens. Its annotations are grounded in clinical expertise, ensuring that labels are precise, consistent, and highly relevant to medical practice.

The second dataset, the **Medical Abstracts Corpus** [2], is publicly available on GitHub at `sebischair/Medical-Abstracts-TC-Corpus` and on Hugging Face as the Medical Abstracts Dataset. It consists of medical abstracts categorized by pathological disease types. Although it contains fewer tokens than the MSKCC dataset, it provides a diverse and carefully curated collection of clinically relevant texts. Its sources—peer-reviewed literature and standardized indexing systems—support the accuracy and consistency of its annotations, making it a reliable benchmark for biomedical NLP research.

3.2 Datasets Overview and Statistics

The first dataset **Memorial Sloan Kettering Cancer Center (MSK)** was utilized in a challenge aiming to develop models capable of *classifying genetic mutations* based on *clinical textual evidence* drawn from medical reports, scientific literature, and other clinical documents.

The dataset comprises two main files:

Table 3.1: Structure of the MSK Kaggle Challenge Dataset

Variable	Description
`ID`	Unique identifier for linking with clinical evidence (in `training_variants`) or matching the mutation metadata (in `training_text`).
`Gene`	Gene in which the mutation occurs.
`Variation`	Amino acid change caused by the mutation.
`Class`	Integer (1–9) indicating the clinical relevance category.
`Text`	Clinical description and literature excerpts used for classification.

In order to enhance the predictive power of our models, we created a new variable `abstract` by concatenating the `Gene`, `Variation`, and `Text` fields for each mutation instance: This combined textual representation captures both the gene-specific information and the clinical context of the mutation,

[1] `https://www.kaggle.com/competitions/msk-redefining-cancer-treatment`

[2] `https://github.com/sebischair/Medical-Abstracts-TC-Corpus`

serving as the primary input for our text classification experiments. By using this aggregated field, we aim to provide the model with richer and more informative features, which improve class separability while maintaining all relevant biological and clinical information.

The dataset contains approximately *3,400* abstracts related to genetic mutations and their clinical implications.

Originally a **9-class** problem, the dataset's label distribution is shown in Table 3.2. Each class corresponds to a specific mutation type, as defined by the original challenge.

Table 3.2: Class distribution in the MSK Kaggle Challenge Dataset

Class ID	Mutation Type	Number of Samples
1	Mutation1	566
2	Mutation2	452
3	Mutation3	89
4	Mutation4	686
5	Mutation5	242
6	Mutation6	273
7	Mutation7	952
8	Mutation8	19
9	Mutation9	37

The abstracts range from *3* to *4,870* sentences each, with an average of *567.56* sentences per abstract, amounting to a total of *1,881,021* sentences across the dataset.

Similarly, Tables 3.3 and 3.4 provide the total sentence and word counts per category, respectively, with individual abstracts ranging from a minimum of *53* words to a maximum of *76,782* words, and an average of *9,565.51* words per abstract.

Table 3.3: Total sentence count per category

Mutation Type	Total Sentences
Mutation8	11,955
Mutation9	26,050
Mutation3	34,925
Mutation5	109,264
Mutation6	144,978
Mutation2	241,087
Mutation1	317,519
Mutation4	365,724
Mutation7	**630,519**

Table 3.4: Total word count per category

Mutation Type	Total Words
Mutation8	205,474
Mutation9	473,876
Mutation3	601,407
Mutation5	1,819,126
Mutation6	1,975,631
Mutation2	4,208,298
Mutation1	5,367,768
Mutation4	6,162,606
Mutation7	**10,905,052**

To better understand the distribution of sentence and word counts across mutation types, we employed boxplots and histograms (figures 3.1–3.4). These visualizations provide an intuitive overview of the variability, skewness, and presence of outliers in the dataset, thereby complementing the tabulated statistics.

For our experiments, we reformulated the task as a **binary classification** problem: *Positive Class (1):* Class 7 (*Mutation7*) – 952 samples, and *Negative Class (0):* All other classes (1–6, 8, 9) – 2,364 samples.

Tables 3.5 and 3.6 summarize the total sentence and word counts for each binary class, while figures 3.6 and 3.5 illustrate their distributions across abstracts.

This reformulation aims to create a binary classification problem and allows us to conduct our experiments efficiently, avoiding memory issues on Google Colab when applying SMOTE to the data.

Table 3.5: Total sentence count per binary class

Binary Class	Total Sentences
0 (Negative)	**1,251,502**
1 (Positive)	630,519

Table 3.6: Total word count per binary class

Binary Class	Total Words
0 (Negative)	**20,814,186**
1 (Positive)	10,905,052

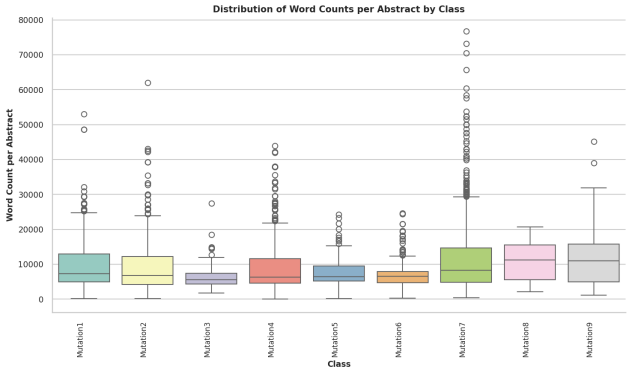


Figure 3.1: Boxplot of word counts per abstract by mutation type

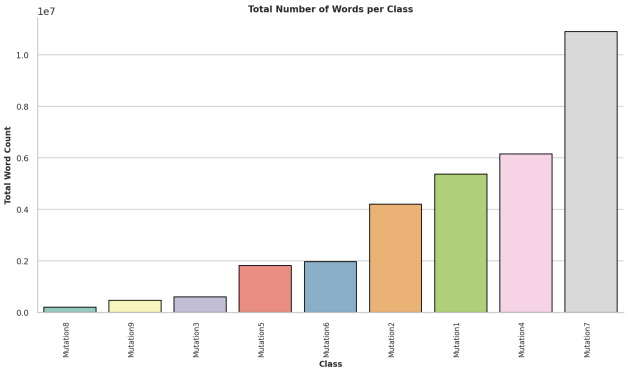


Figure 3.2: Histogram of word count distribution across abstracts

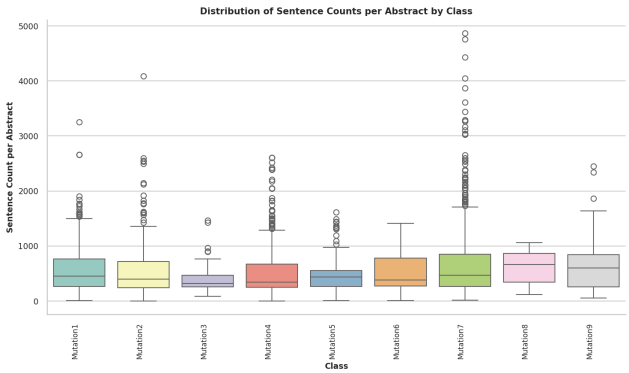


Figure 3.3: Boxplot of sentence counts per abstract by mutation type

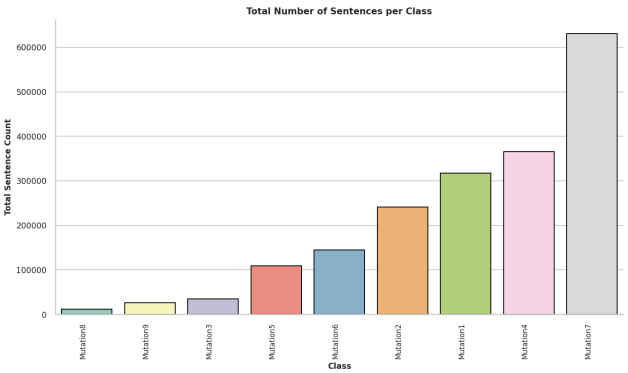


Figure 3.4: Histogram of sentence count distribution across abstracts

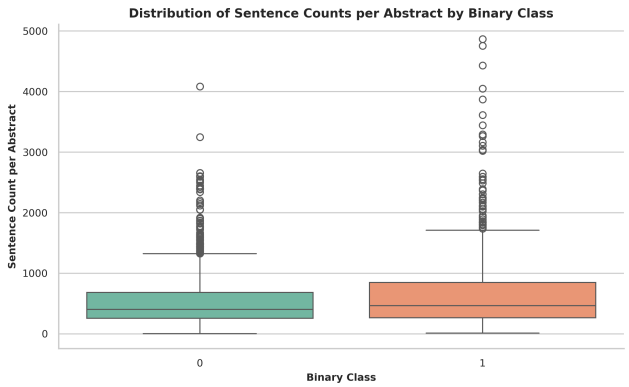


Figure 3.5: Distribution of sentence counts per abstract by binary class

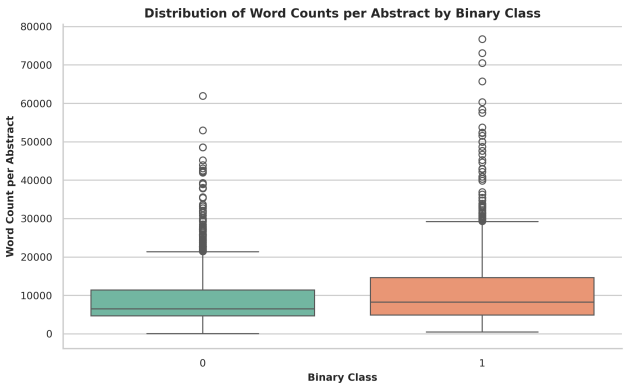


Figure 3.6: Distribution of word counts per abstract by binary class

3.2.1 Medical Abstracts Dataset

Regarding the Medical Abstracts Dataset, it contains textual summaries of patient conditions, each labeled according to one of five categories. The dataset is structured with two main variables: the category label and the corresponding medical abstract. Table 3.7 provides a clear description of these variables.

Table 3.7: Structure of the Medical Abstracts Dataset

Variable	Description
`condition_label`	Integer (1–5) representing the patient condition category: 1 = Neoplasms, 2 = Digestive system diseases, 3 = Nervous system diseases, 4 = Cardiovascular diseases, 5 = General pathological conditions.
`medical_abstract`	Text of the medical abstract summarizing clinical information related to the patient condition.

The abstracts range from *2* to *54* sentences each, with an average of *10.71* sentences per abstract, amounting to a total of *123,673* sentences across the dataset.

Tables 3.9 and 3.10 present the total sentence and word counts per category, respectively. Individual abstracts range from a minimum of *24* words to a maximum of *596* words, with an average of *179.64* words per abstract. For this dataset, we retained the original **5-class** structure to perform **multi-class text classification** experiments. Figures 3.7–3.10 offer a visual summary of the dataset's variability, skewness, and potential outliers, serving as a complementary perspective to the tabulated statistics.

Table 3.8: Category distribution in the Medical Abstracts Dataset

Label ID	Category	Number of Abstracts
1	Neoplasms	2,530
2	Digestive system diseases	1,195
3	Nervous system diseases	1,540
4	Cardiovascular diseases	2,441
5	General pathological conditions	**3,844**

Table 3.9: Total sentence count per category

Category	Total Sentences
Digestive system diseases	12,980
Nervous system diseases	14,401
Neoplasms	25,377
Cardiovascular diseases	30,126
General pathological conditions	**40,789**

Table 3.10: Total word count per category

Category	Total Words
Digestive system diseases	213,502
Nervous system diseases	251,541
Neoplasms	440,660
Cardiovascular diseases	488,529
General pathological conditions	**680,617**

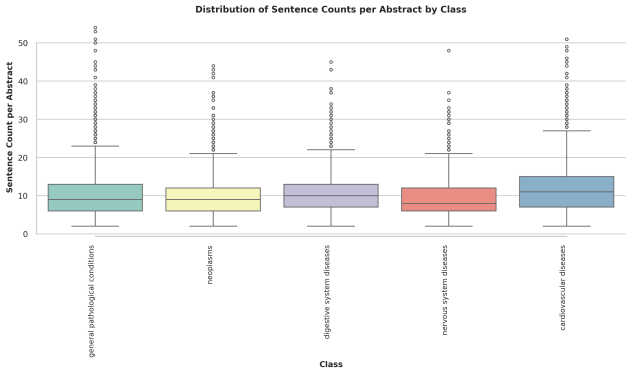


Figure 3.7: Boxplot of sentence counts per abstract by category

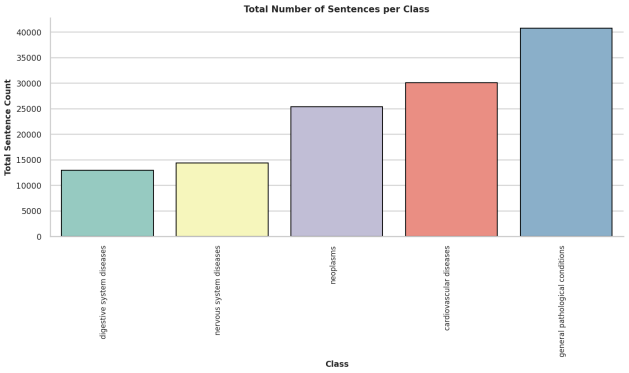


Figure 3.8: Histogram of sentence counts per abstract by category

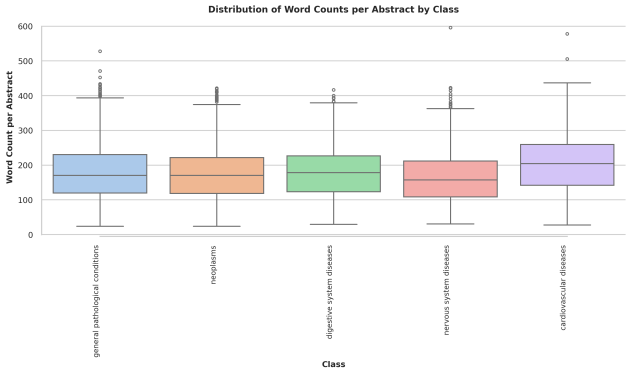


Figure 3.9: Boxplot of word counts per abstract by category

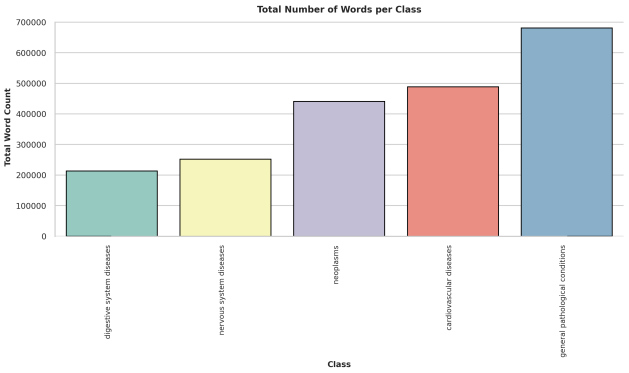


Figure 3.10: Histogram of word counts per abstract by category

3.2.2 t-SNE Visualizations

These t-SNE plots 3.11–3.14 and plots 3.15–3.18) respectively represent the embeddings of the MSKCC and Medical Abstracts datasets before any model training. Since the embeddings are high-dimensional, direct visualization is not possible. We therefore use t-SNE to project them into two dimensions, which allows us to assess the intrinsic structure of the data and the separability of different classes. Each point corresponds to a genetic mutation instance, colored according to its class label.

Several observations can be made from these visualizations:

- The *t-SNE from scratch* plot shows the raw embeddings without any pre-trained knowledge. Classes are partially intermixed, indicating limited separability based solely on initial feature representations.
- Using *GloVe embeddings* provides a modest improvement in clustering, as seen by slightly more compact class regions.
- *FastText embeddings* capture subword information, which improves the grouping of instances with similar morphological patterns.
- *BioMedNLP embeddings* exhibit the clearest separation among classes, highlighting the benefits of domain-specific embeddings for biomedical text representation.

It is important to note that the observed distributions are strongly influenced by the number of observations in each dataset. Consequently, the t-SNE plots for the MSKCC dataset show fewer clusters due to its smaller number of observations (around 3,400), while the Medical Abstracts dataset forms a large cluster because of its much larger number of abstracts. However, this difference in visual clustering does not reduce the complexity of the MSKCC dataset, which contains a significantly higher number of tokens per instance compared to the Medical Abstracts dataset.

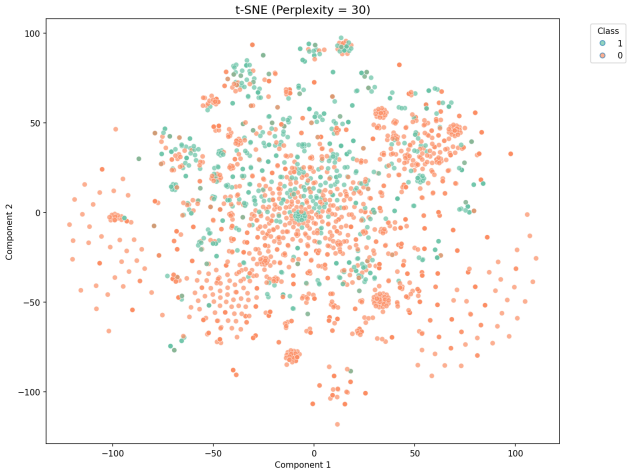


Figure 3.11: t-SNE with raw text

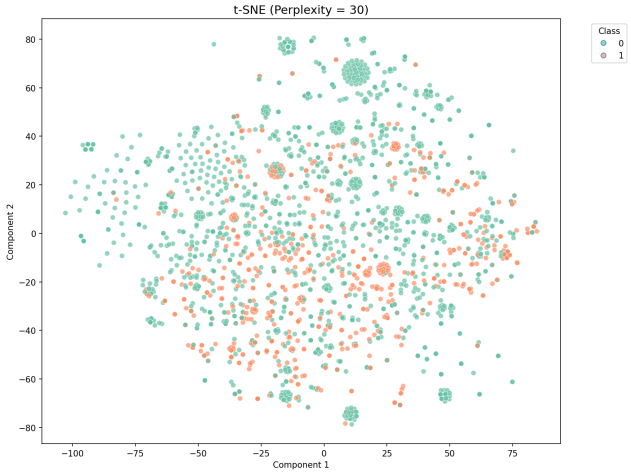


Figure 3.12: t-SNE with GloVe embeddings

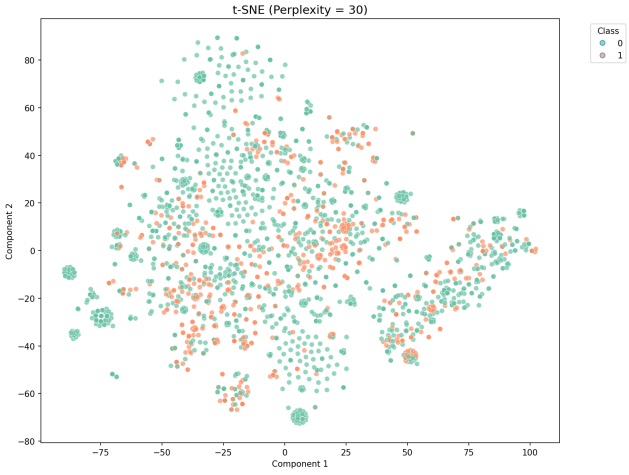


Figure 3.13: t-SNE with FastText embeddings

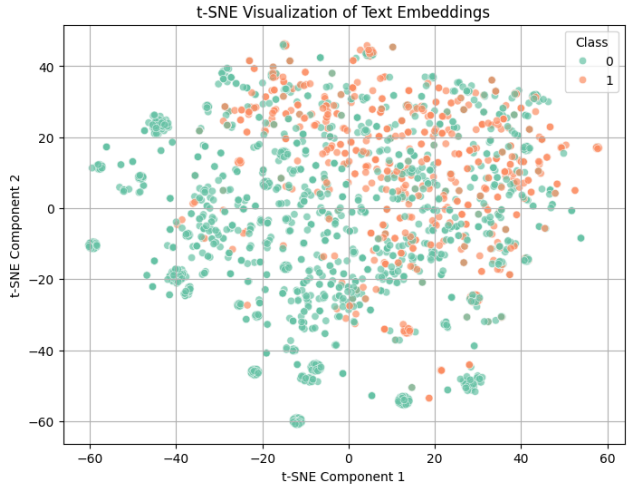


Figure 3.14: t-SNE with BioMedNLP embeddings

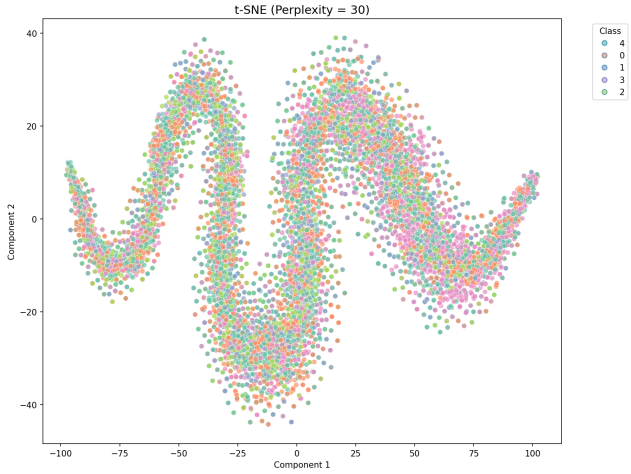


Figure 3.15: t-SNE with raw text

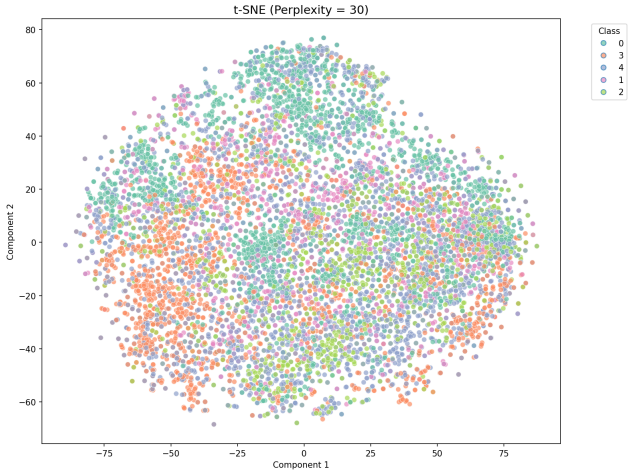


Figure 3.16: t-SNE with GloVe embeddings

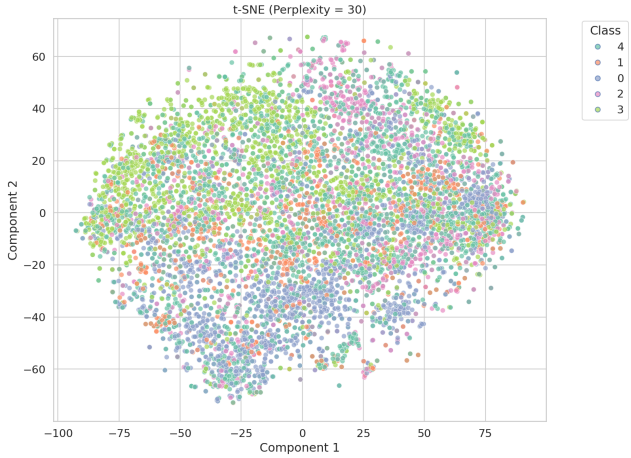


Figure 3.17: t-SNE with FastText embeddings

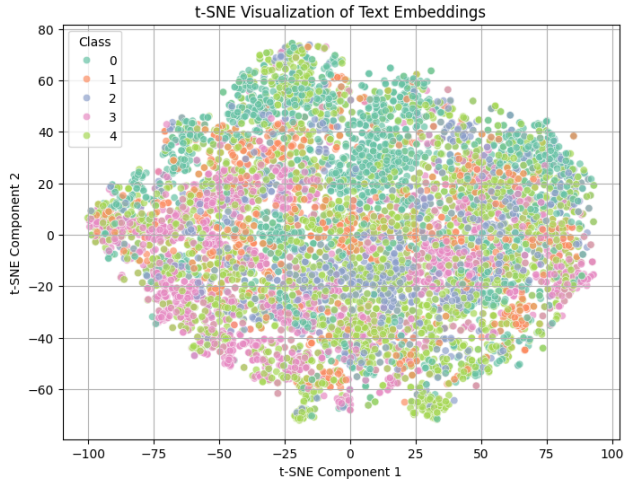


Figure 3.18: t-SNE with BioMedNLP embeddings

3.3 Conclusion

The two datasets analyzed exhibit distinct characteristics and structures. The MSKCC dataset focuses on predicting genetic mutations from clinical abstracts, while the Medical Abstracts Dataset covers multiple categories of diseases. Detailed statistical analyses and t-SNE visualizations reveal the variability, distribution, class separability, and the presence of class imbalance within each dataset. Together, these datasets provide a solid foundation for developing and evaluating biomedical text classification models, whether for binary or multi-class tasks.

Chapter 4

Methodological Approach and Network Architectures

4.1 Model Architectures and Embedding Techniques

The models evaluated in this work are based on bidirectional recurrent architectures, namely LSTM and GRU. Each model begins with an embedding layer, which is either learned during training or initialized with pre-trained vectors, including GloVe (300d), FastText (300d), or BioMedNLP contextual embeddings. To further enhance predictive performance, we incorporate Bahdanau attention on top of the recurrent layers.

4.2 Training Methodology

The model training followed a multi-step methodology inspired by standard practices for evaluating machine learning models. This approach includes the following steps:

1. *Model training*: All models were trained on the training dataset using different algorithms and/or varying complexity values. To enhance stability during training—particularly for recurrent architectures such as LSTM and GRU—**gradient clipping** was applied to constrain the norm of the gradients, effectively preventing exploding gradient issues. In addition, **dropout** regularization was employed within the architectures to reduce the risk of overfitting by randomly deactivating a subset of neurons during each training iteration. **Early Stopping** (ES) was also implemented, monitoring the validation loss during training and halting the process when performance no longer improved, thus preventing overfitting and saving computational resources.

2. *Best model selection*: After the initial training, the model that achieved the best performance on the validation set was selected. This step enables choosing the most effective model while avoiding overfitting by using a separate validation set.

3. *Hyperparameter optimization with Grid Search*: Following model selection, a hyperparameter optimization phase was conducted using the *Grid Search* method. This method performs an exhaustive search over a predefined set of hyperparameters for each model to maximize performance. Using the validation set, Grid Search explores all possible hyperparameter combinations to find the best configuration, thus ensuring optimal model performance.

4. *Retraining on the full dataset*: Once the best hyperparameters were identified via Grid Search, the model was retrained using the entire available dataset, i.e., the combination of the training and validation sets. This step maximizes data utilization to improve model performance.

5. *Testing on the test set*: After retraining, the model was tested on a separate test set to evaluate its ability to generalize to new data, providing an accurate estimate of model performance.

6. *Final retraining*: Finally, after the final evaluation, the model was retrained one last time using all available data. This last step aims to make the best use of the data before applying the model to new datasets or real-world scenarios.

This methodology ensures that the model is not only performant but also robust and capable of generalizing to unseen data by following a rigorous training and validation process. The use of gradient clipping, dropout, and early stopping further strengthens the training procedure by enhancing numerical stability, reducing overfitting, and avoiding unnecessary training iterations.

For all experiments, the loss function used was *cross entropy* for multi-class classification tasks and *binary cross entropy* for binary classification tasks.

4.3 Evaluation Metrics

The classification performance was assessed using a variety of complementary metrics. *Balanced accuracy* was chosen to account for class imbalance by averaging the model's recall across all categories, ensuring that each class contributed equally to the overall score. *Precision* and *recall* were used to measure, respectively, the relevance of the predicted positives and the model's ability to correctly identify all actual positives. Their combination, the *F1-score*, provided a balanced indicator that considers both correctness and completeness of the predictions. Additionally, the *AUC-ROC* metric was used to evaluate the model's ability to distinguish between different classes, with higher values indicating better discriminative performance.

Balanced accuracy was chosen to account for class imbalance. For binary classification, it is defined as:

$$\textit{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

where TP, TN, FP, and FN represent true positives, true negatives, false positives, and false negatives, respectively. For multi-class classification, balanced accuracy is computed as the average recall across all classes C:

$$\textit{Balanced Accuracy} = \frac{1}{|C|}\sum_{i \in C} \frac{TP_i}{TP_i + FN_i}$$

Precision and *recall* measure, respectively, the relevance of predicted positives and the model's ability to correctly identify all actual positives:

$$\textit{Precision} = \frac{TP}{TP+FP}, \quad \textit{Recall} = \frac{TP}{TP+FN}$$

The *F1-score* is the harmonic mean of precision and recall:

$$\textit{F1-score} = 2 \cdot \frac{\textit{Precision} \cdot \textit{Recall}}{\textit{Precision} + \textit{Recall}}$$

For multi-class classification, these metrics are computed per class i and averaged (macro-average) across all classes:

$$\textit{Precision}_{macro} = \frac{1}{|C|}\sum_{i \in C} \textit{Precision}_i, \quad \textit{Recall}_{macro} = \frac{1}{|C|}\sum_{i \in C} \text{Recall}_i, \quad F1_{macro} = \frac{1}{|C|}\sum_{i \in C} F1_i$$

Additionally, the *AUC* metric was used to evaluate the model's ability to distinguish between different classes. For binary classification, it corresponds to the area under the curve plotting the true positive rate against the false positive rate:

$$\text{AUC-ROC} = \int_0^1 TPR(FPR)\, dFPR$$

For multi-class problems, a one-vs-rest strategy was used to compute the AUC for each class, and then the results were averaged.

Chapter 5

Experiments

This chapter presents the outcomes obtained during the test phase of various experiments. Due to class imbalance, the following techniques were applied: `SMOTE` (Synthetic Minority Oversampling Technique) for data augmentation, and `Class weighting` in the loss function to mitigate bias toward majority classes. The models, which include both binary and multi-class classification architectures, were trained using either `Cross Entropy` loss (for multi-class tasks) or `Binary Cross Entropy` loss (for binary classification tasks). Model performance was evaluated using several metrics: *Balanced Accuracy*, *F1-score*, *Precision*, *Recall*, and *AUC*.

5.1 Experiments on the Memorial Sloan Kettering Cancer Center (MSKCC) Dataset

The Memorial Sloan Kettering Cancer Center (MSK) – Kaggle Challenge Dataset contains clinical and biomedical text samples provided as part of a public competition hosted on Kaggle. The dataset was partitioned into separate training (70%), validation (15%), and test (15%) sets to ensure consistent evaluation across models. In total, the dataset consisted of 3,316 samples, with 2,321 samples used for training, 497 for validation, and 498 reserved for independent testing.

5.1.1 Experiment with Machine Learning Models — Linear SVM and Random Forest

In this first experiment, we established baseline results using classical machine learning algorithms before moving to deep learning architectures such as LSTM, GRU, and Bahdanau attention. The biomedical abstracts from the MSKCC dataset were represented using a `TF-IDF` vectorization approach with a maximum of 5000 features and an n-gram range of (1, 2). This representation transforms the raw text into sparse numerical vectors that can be directly processed by machine learning models.

Experiment 1 : Without applying SMOTE

The models were trained without any resampling technique in order to assess their performance under the natural class imbalance of the dataset. Table 5.1 summarizes the pipeline configurations, while Table 5.2 reports the obtained performance metrics.

Table 5.1: Machine Learning Pipeline Configurations for MSK Dataset

Model	**Pipeline Components and Parameters**
Linear SVM	`TfidfVectorizer`: max_features=5000, ngram_range=(1, 2) `SVC`: C=0.1, kernel='linear', gamma='scale', class_weight='balanced', probability=True
Random Forest	`TfidfVectorizer`: max_features=5000, ngram_range=(1, 2) `RandomForestClassifier`: max_depth=10, n_estimators=200, min_samples_split=2, class_weight='balanced', random_state=42

Table 5.2: Performance Metrics for Machine Learning Models on MSK Dataset Without SMOTE

Model	Bal. Acc.	F1-score	Recall	Precision	AUC
Linear SVM	0.818	0.693	0.965	0.541	0.0885
Random Forest	0.781	0.700	0.629	0.789	0.925

The results show that the Linear SVM achieves a higher recall (0.965), meaning it identifies most positive cases but at the expense of precision (0.541), while the Random Forest exhibits more balanced precision and recall values.

Experiment 2 : with SMOTE applied

In this setting, the Synthetic Minority Oversampling Technique (SMOTE) was applied to balance the classes before training the models. Table 5.1 recalls the pipeline configurations, while Table 5.3 reports the evaluation metrics obtained after applying SMOTE to the training set.

Table 5.3: Performance Metrics for Machine Learning Models on MSK Dataset With SMOTE

Model	Bal. Acc.	F1-score	Recall	Precision	AUC
Linear SVM	0.816	0.694	0.944	0.549	0.0889
Random Forest	0.819	0.746	0.727	0.765	0.917

Conclusion

Compared to the results obtained without SMOTE (Table 5.2), the impact of class balancing is evident. For the Linear SVM, performance remains relatively stable, with only minor fluctuations in recall (from 0.965 to 0.944) and precision (from 0.541 to 0.549). The AUC values for SVM remain very low in both cases (0.0885 without SMOTE vs. 0.0889 with SMOTE), indicating that the model struggles to discriminate between classes regardless of oversampling.

In contrast, the Random Forest benefits more significantly from SMOTE: both balanced accuracy (from 0.781 to 0.819) and F1-score (from 0.700 to 0.746) improve, with a more favorable trade-off between recall ($0.629 \rightarrow 0.727$) and precision ($0.789 \rightarrow 0.765$). The AUC remains high (0.925 without SMOTE vs. 0.917 with SMOTE), reflecting strong overall discrimination capability, though slightly reduced after oversampling, which may indicate a minor shift in the decision threshold due to balancing.

These results suggest that oversampling techniques such as SMOTE can effectively mitigate the impact of class imbalance, particularly for ensemble-based classifiers, while linear models like SVM are less sensitive to this adjustment.

To further illustrate these findings, Figures 5.1–5.6 present the ROC curves and confusion matrices for both models, with and without SMOTE applied.

Building on these insights from classical machine learning models, we now turn our attention to more sophisticated sequence modeling approaches. Deep learning architectures, particularly recurrent neural networks and their variants, have the potential to capture complex patterns and contextual dependencies in text that traditional models may overlook. In the following section, we describe experiments with LSTM- and GRU-based models, both trained from scratch and initialized with pre-trained embeddings, and further enhanced with attention mechanisms and contextualized biomedical embeddings.

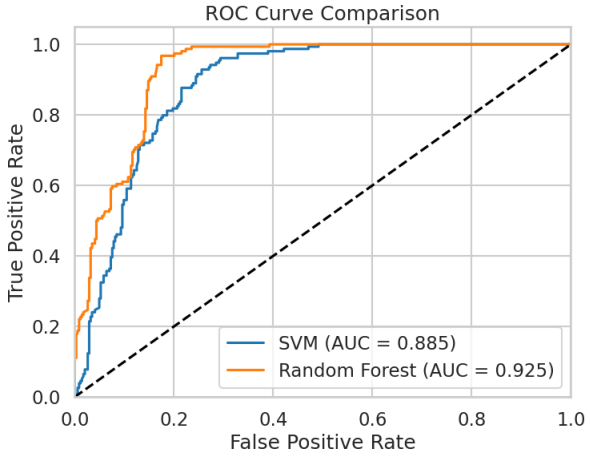


Figure 5.1: ROC curve without SMOTE

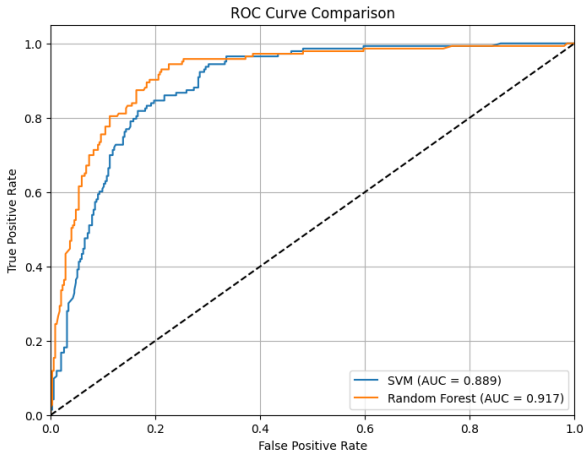


Figure 5.2: ROC curve with SMOTE

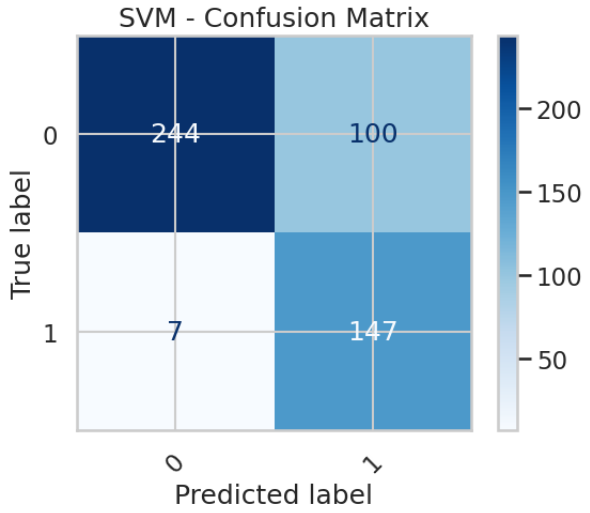


Figure 5.3: Confusion matrix of SVM without SMOTE

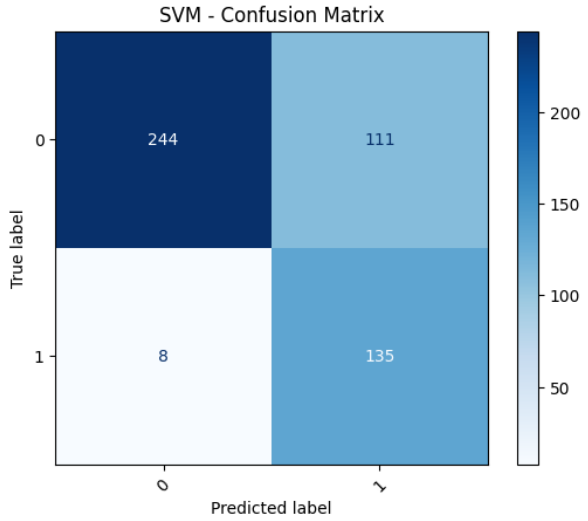


Figure 5.4: Confusion matrix of SVM with SMOTE

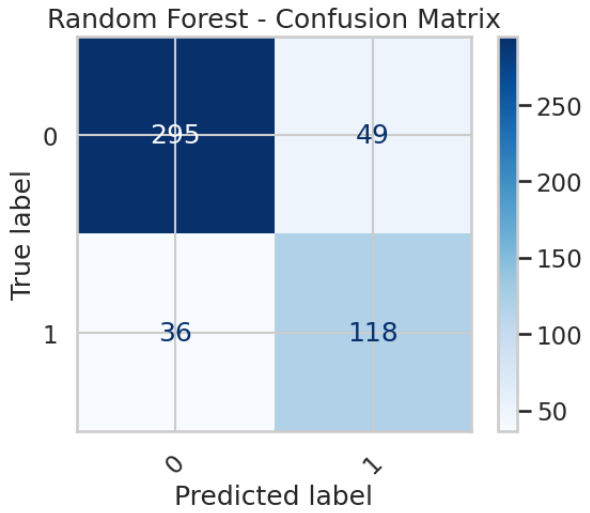


Figure 5.5: Confusion matrix of Random Forest without SMOTE

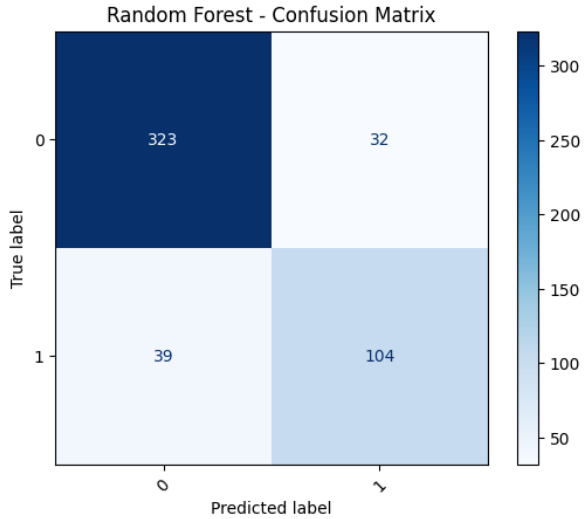


Figure 5.6: Confusion matrix of Random Forest with SMOTE

5.1.2 Experiment with Deep Learning Models

The vocabulary was constructed by tokenizing all training texts and removing tokens with a frequency lower than 3, resulting in a vocabulary size of *40,865* unique tokens. In this part, we begin with LSTM- and GRU-based models trained from scratch, followed by models initialized with pre-trained word embeddings (GloVe (300d) and Fasttext (300d)). Next, we incorporate an attention mechanism (Bahdanau attention) to enhance sequence modeling, and finally, we evaluate models using BioMedNLP contextual embeddings.

During the model selection and optimization phases [1], we systematically explored key hyperparameters, including optimizer choice (Adam, RMSprop, and AdamW), batch size (8, 16, and 32), dropout rates, and hidden layer dimensions.

For the purpose of clarity and conciseness, in the following sections we will primarily present the test results and a few key plots obtained during the optimization phase. Everything related to the detailed optimization process is fully documented in the GitHub repository mentioned above.

5.1.2.1 Experiment with model-based LSTM and GRU with embeddings learned from scratch

Table 5.4 summarizes the model configurations, hyperparameters, batch size, and optimizers used for the LSTM and GRU models trained from scratch. These same configurations were consistently applied across all subsequent experiments, including those with SMOTE and class weighting.

Table 5.4: Model configurations, hyperparameters, batch size and optimizers

Model	Embedding Dim.	Hidden Dim.	Dropout	Total Params	Optimizer & LR
LSTM	450	500	0.2	22,199,151	RMSprop, 1×10^{-3}
GRU	350	400	0.2	16,109,051	RMSprop, 1×10^{-3}

Experiment 1 : Without applying SMOTE or Class Weighting

Table 5.5 presents the evaluation metrics, early stopping epochs, and average training time per epoch for the models trained without SMOTE or class weighting.

Table 5.5: Performance metrics and training details without applying SMOTE or class weighting

Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	Bal. Acc.	F1	Prec.	Recall
LSTM	13	0.27	0.6643	75.10%	71.93%	61.25%	59.04%	63.64%
GRU	17	0.20	0.6480	77.31%	72.64%	62.21%	64.14%	60.39%

Experiment 2 : With SMOTE applied

Table 5.6 shows the performance of the same LSTM and GRU models when SMOTE oversampling was applied.

Table 5.6: Performance of LSTM and GRU models with SMOTE applied

Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
LSTM	13	0.44	0.8283	74.70%	61.11%	71.82%	58.24%	64.29%
GRU	19	0.29	0.6538	79.32%	65.55%	74.99%	67.59%	63.64%

Experiment 3 : With Class Weighting applied

Table 5.7 summarizes the results when class weighting was applied, again using the same model configurations.

[1] `https://github.com/etoMvom/biomedical_text_classification`

Table 5.7: Performance of LSTM and GRU models with class weighting applied

Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
LSTM	9	0.29	0.9161	76.31%	63.80%	73.88%	60.47%	67.53%
GRU	14	0.17	0.9656	78.51%	67.08%	76.38%	63.74%	70.78%

Conclusion

Across all the previous experiments with SMOTE and with class weighting—the LSTM and GRU models were trained using the same configurations, including embedding dimension, hidden dimension, dropout, batch size, and optimizer settings, ensuring that performance differences are attributable primarily to data handling strategies rather than model architecture. GRU models consistently achieved slightly higher test accuracy and balanced accuracy than LSTM models under identical settings. The application of SMOTE generally improved F1-score and balanced accuracy, indicating better handling of class imbalance, while class weighting further enhanced performance on minority classes, particularly in F1, precision, and recall metrics. When compared to the classical machine learning models previously evaluated, both LSTM and GRU outperformed Linear SVM in balanced accuracy and F1-score, highlighting their ability to capture complex sequential dependencies in text data. Although Random Forest benefited substantially from SMOTE, the recurrent neural networks demonstrated higher overall balanced accuracy and a more favorable trade-off between recall and precision.

5.1.2.2 Experiment with model-based LSTM and GRU using pre-trained embeddings (GloVe-300d and FastText-300d)

Tables 5.8 summarizes the model configurations, hyperparameters, and optimizers used for the LSTM and GRU models with pre-trained embeddings. The batch size was set to 16 and the depth was set to 1. **These same configurations were consistently applied across all subsequent experiments**, including those with SMOTE and class weighting.

Table 5.8: Model configurations, hyperparameters, and optimizers with pre-trained embeddings

Embedding	Model	Embedding Dim.	Hidden Dim.	Dropout	Total Params	Optimizer & LR
GloVe-300d	LSTM	300	356	0.5	14,134,797	RMSprop, 1×10^{-3}
GloVe-300d	GRU	300	500	0.3	14,667,101	RMSprop, 1×10^{-3}
FastText-300d	LSTM	300	356	0.5	14,134,797	RMSprop, 1×10^{-3}
FastText-300d	GRU	300	500	0.3	14,667,101	RMSprop, 1×10^{-3}

Experiment 1: Without applying SMOTE or Class Weighting

Table 5.9 presents the evaluation metrics, early stopping epochs, and average training time per epoch for models trained without SMOTE or class weighting.

Table 5.9: Performance of LSTM and GRU models without SMOTE or class weighting

Embedding	Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
GloVe-300d	LSTM	16	0.16	0.5833	75.70%	65.92%	75.78%	58.21%	75.97%
GloVe-300d	GRU	20	0.20	0.6600	77.51%	65.85%	75.47%	62.07%	70.13%
FastText-300d	LSTM	11	0.16	0.5709	73.69%	61.58%	72.17%	56.15%	68.18%
FastText-300d	GRU	18	0.20	0.6372	79.12%	67.50%	76.63%	65.06%	70.13%

Experiment 2: With SMOTE applied

Table 5.10 shows the performance of the same models when SMOTE oversampling was applied.

Table 5.10: Performance of LSTM and GRU models with SMOTE applied

Embedding	Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
GloVe-300d	LSTM	20	0.22	0.8719	75.10%	64.77%	74.80%	57.58%	74.03%
GloVe-300d	GRU	10	0.31	0.5481	77.91%	68.93%	78.27%	61.00%	79.22%
FastText-300d	LSTM	19	0.23	0.7493	76.10%	67.22%	76.97%	58.37%	79.22%
FastText-300d	GRU	16	0.28	0.6000	80.52%	71.22%	79.80%	65.57%	77.92%

Experiment 3: With Class Weighting applied

Table 5.11 summarizes the results when class weighting was applied.

Table 5.11: Performance of LSTM and GRU models with class weighting applied

Embedding	Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
GloVe-300d	LSTM	30	0.16	1.6540	73.90%	63.89%	74.11%	55.83%	74.68%
GloVe-300d	GRU	22	0.20	0.7243	76.51%	69.45%	79.23%	58.08%	86.36%
FastText-300d	LSTM	20	0.16	1.2965	75.70%	65.92%	75.78%	58.21%	75.97%
FastText-300d	GRU	12	0.20	0.7397	78.92%	71.07%	80.26%	61.72%	83.77%

Conclusion

Compared to models using embeddings learned from scratch, GRU models demonstrated superior test accuracy and balanced accuracy over LSTM models under the same conditions. Applying SMOTE further enhanced F1-score and balanced accuracy, while class weighting provided additional gains for minority class performance.

5.1.2.3 Model-based LSTM and GRU with Bahdanau Attention using Word Embeddings

We trained LSTM and GRU models with Bahdanau attention using 300-dimensional GloVe and FastText embeddings. The same model configurations 5.12 were kept consistent across all experiments to ensure a fair comparison of results. The models were trained using the RMSprop optimizer with a batch size of 16, a depth of 1, and learning rates set to 10^{-3} for LSTM and 10^{-4} for GRU.

Table 5.12: Model configuration summary

Embedding	Model	Embedding Dim.	Hidden Dim.	Attention	Dropout	Total Params
GloVe-300d	LSTM	300	400	Bahdanau	0.5	14,828,101
GloVe-300d	GRU	300	500	Bahdanau	0.3	14,667,101
FastText-300d	LSTM	300	400	Bahdanau	0.5	14,828,101
FastText-300d	GRU	300	400	Bahdanau	0.5	14,266,501

Experiment 1: Without applying SMOTE or Class Weighting

In this baseline experiment, the models were trained without applying any technique to address class imbalance, allowing us to evaluate their raw performance under the original data distribution.

Table 5.13: Performance of LSTM and GRU models without SMOTE or class weighting

Embedding	Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
GloVe-300d	LSTM	10	0.22	0.4100	83.13%	76.80%	85.10%	66.83%	90.26%
GloVe-300d	GRU	19	0.20	0.6104	77.11%	66.86%	76.44%	60.53%	74.68%
FastText-300d	LSTM	16	0.24	0.6726	78.92%	65.80%	75.23%	66.01%	65.58%
FastText-300d	GRU	12	0.18	0.3831	83.13%	75.58%	83.49%	68.42%	84.42%

Experiment 2: With SMOTE applied

Afterwards in this experiment, the Synthetic Minority Oversampling Technique (SMOTE) was applied to the training data to mitigate class imbalance by generating synthetic examples for minority classes.

Table 5.14: Performance of LSTM and GRU models with SMOTE applied

Embedding	Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
GloVe-300d	LSTM	15	0.20	0.5247	81.93%	73.53%	81.72%	67.20%	81.17%
GloVe-300d	GRU	12	0.24	0.4340	81.73%	73.78%	82.11%	66.32%	83.12%
FastText-300d	LSTM	14	0.21	0.5123	82.10%	74.05%	81.90%	67.85%	81.50%
FastText-300d	GRU	13	0.25	0.4255	81.95%	73.95%	82.35%	66.80%	83.50%

Experiment 3: With Class Weighting applied

And finally in this experiment, class weighting was applied during training to penalize misclassification of minority classes more heavily, providing an alternative imbalance handling strategy to compare with SMOTE.

Table 5.15: Performance of LSTM and GRU models with class weighting applied

Embedding	Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
GloVe-300d	LSTM	14	0.22	0.8139	80.52%	74.81%	84.11%	62.34%	93.51%
GloVe-300d	GRU	20	0.20	0.6541	82.73%	75.98%	84.27%	66.67%	88.31%
FastText-300d	LSTM	20	0.24	0.7618	81.73%	74.51%	83.01%	65.52%	86.36%
FastText-300d	GRU	11	0.18	0.5445	81.73%	75.86%	84.80%	64.13%	92.86%

Conclusion

In all experimental setups, the integration of Bahdanau attention consistently improved the performance of both LSTM and GRU models compared to their non-attention counterparts. The attention mechanism enhanced interpretability and allowed the models to focus more effectively on salient parts of the input sequences, resulting in stronger recall and F1-scores.

Without any imbalance handling, the LSTM with GloVe-300d achieved the highest recall (90.26%), while the GRU with FastText-300d balanced high accuracy (83.13%) and robust F1 performance. When SMOTE was applied, results stabilized for both models, with LSTM and GRU reaching balanced accuracy above 81%, showing that oversampling complemented the attention mechanism in mitigating class bias.

Class weighting further amplified minority class detection, especially in GRU-based models, which achieved the strongest trade-off between precision and recall. Notably, class-weighted training yielded the highest recall (92.86% with FastText-GRU) and F1 gains in all tested conditions, though sometimes at the expense of higher loss values.

5.1.2.4 Experiment with Model-based LSTM and GRU with Bahdanau Attention using BioMedNLP Contextual Embeddings

We evaluate LSTM and GRU models with Bahdanau attention using BioMedNLP contextual embeddings. The experiments explore three handling strategies: no SMOTE or class weighting, SMOTE, and class weighting. All models are trained with Adam optimizer and learning rate 1×10^{-4} unless otherwise stated.

Experiment 1: Without SMOTE or Class Weighting

Table 5.16: Model configurations for LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (without SMOTE and class weighting)

Model	Hidden Dim.	Dropout	Attention Dim.	Total Params	Optimizer & LR
LSTM	286	0.3	286	2,580,865	Adam, 1×10^{-4}
GRU	286	0.3	286	1,976,833	Adam, 1×10^{-4}

Table 5.17: Performance of LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (without SMOTE and class weighting)

Model	Epoch	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Precision	Recall
LSTM	15	1.87	0.3519	84.34%	78.09%	85.97%	68.81%	90.26%
GRU	9	1.67	0.3866	81.12%	71.34%	79.70%	67.24%	75.97%

Experiment 2: With SMOTE

Table 5.18: Model configurations for LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (with SMOTE)

Model	Hidden Dim.	Dropout	Attention Dim.	Total Params	Optimizer & LR
LSTM	256	0.8	256	2,233,345	Adam, 1×10^{-4}
GRU	286	0.3	286	1,976,833	Adam, 1×10^{-4}

Table 5.19: Performance of LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (with SMOTE)

Model	Epoch	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Precision	Recall
LSTM	10	2.37	0.4150	79.92%	73.40%	82.60%	62.16%	89.61%
GRU	6	2.34	0.4341	79.12%	71.43%	80.58%	61.90%	84.42%

Experiment 3: With Class Weighting

Table 5.20: Model configurations for LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (with class weighting applied)

Model	Hidden Dim.	Dropout	Attention Dim.	Total Params	Optimizer & LR
LSTM	286	0.3	286	2,580,865	Adam, 1×10^{-4}
GRU	286	0.3	286	1,976,833	Adam, 1×10^{-4}

Table 5.21: Performance of LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (with class weighting)

Model	Epoch	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Precision	Recall
LSTM	9	1.74	0.6044	82.13%	75.62%	84.19%	65.40%	89.61%
GRU	12	1.65	0.5250	84.14%	77.62%	85.47%	68.84%	88.96%

Conclusion

Both LSTM and GRU models with Bahdanau attention and BioMedNLP embeddings achieve strong performance. Without any imbalance handling, LSTM reaches the highest accuracy (84.34%) and balanced accuracy (85.97%), while GRU lags slightly behind. When SMOTE is applied, both models show competitive improvements, with LSTM performing slightly better in F1 score (73.40%). With class weighting, GRU matches and slightly surpasses LSTM in accuracy (84.14% vs. 82.13%) and balanced accuracy (85.47% vs. 84.19%), though LSTM still maintains a strong recall advantage. Overall, BioMedNLP contextual embeddings combined with attention produce robust results, with LSTM leading when trained without imbalance handling, and GRU becoming more effective once class weighting is introduced. Figures 5.7–5.13 present a side-by-side comparison of GRU and LSTM performance using BioMedNLP embeddings across loss, balanced accuracy, confusion matrices, and ROC curves, which highlights these differences and demonstrates the effect of using class weighting to address the class imbalance problem.

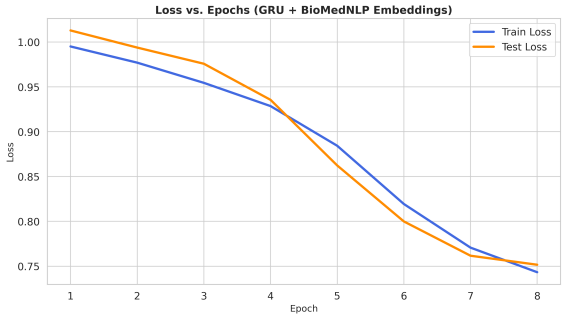


Figure 5.7: GRU + Attention (BioMedNLP) - Loss

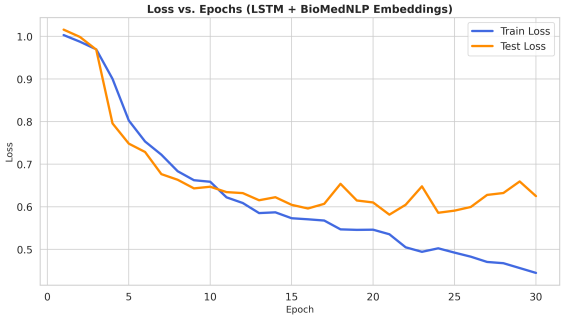


Figure 5.8: LSTM + Attention (BioMedNLP) - Loss

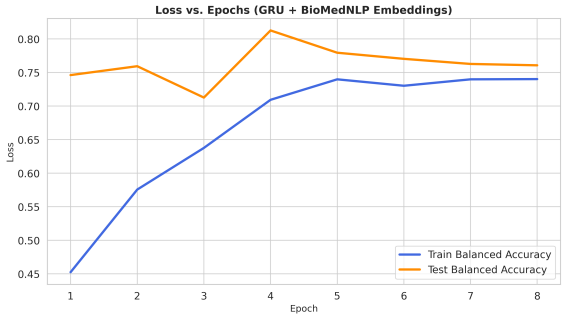


Figure 5.9: GRU + Attention (BioMedNLP) - Balanced Accuracy

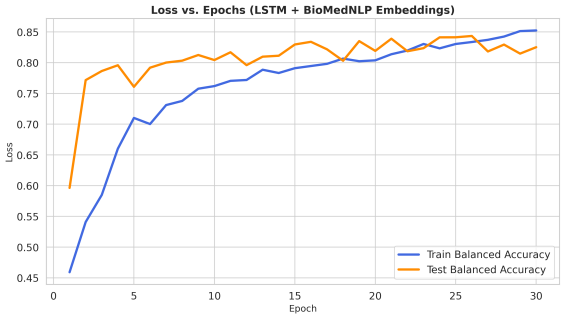


Figure 5.10: LSTM + Attention (BioMedNLP) - Balanced Accuracy

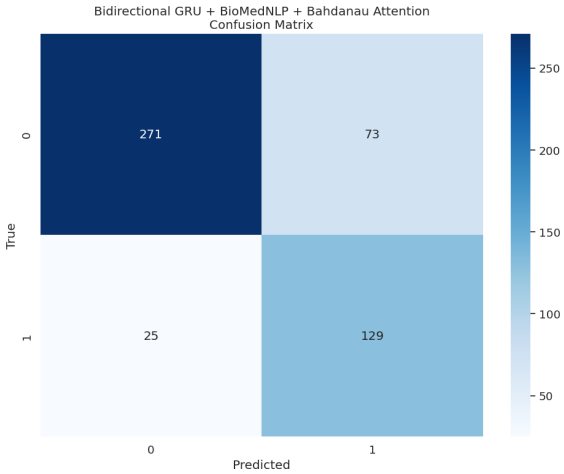


Figure 5.11: GRU + Attention (BioMedNLP) - Confusion Matrix

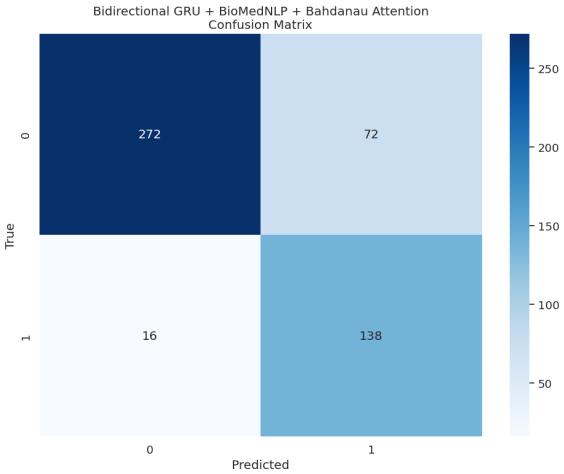


Figure 5.12: LSTM + Attention (BioMedNLP) - Confusion Matrix

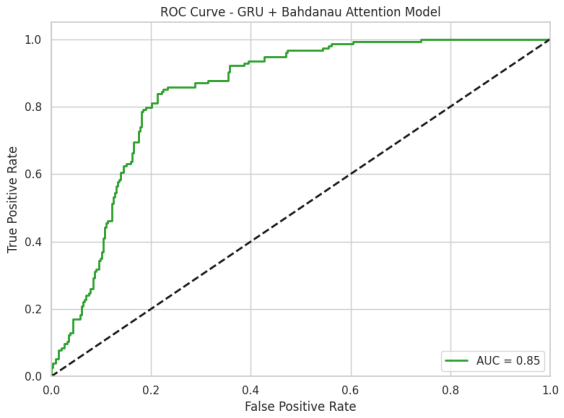


Figure 5.13: GRU + Attention (BioMedNLP) - ROC

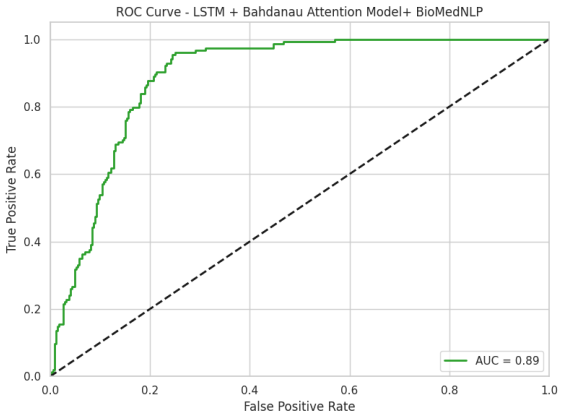


Figure 5.14: LSTM + Attention (BioMedNLP) - ROC

5.1.2.5 Few-Shot Learning on the Memorial Sloan Kettering Cancer Center (MSKCC) Dataset

The few-shot learning experiments were conducted only with models that passed validation and were optimized during model selection. This ensures that the evaluation relies on strong baselines rather than preliminary or underperforming architectures.

It should also be noted that for each experiment, the K abstracts are selected randomly from the training set. Since abstracts exhibit variable token lengths, as shown in the earlier statistical description, this sampling process introduces additional variability. Consequently, the observed fluctuations in performance metrics across different values of K can be partly attributed to this inherent randomness and heterogeneity in input sequences.

Experiment 1: Few-Shot Learning with Model-based LSTM and GRU using Embeddings Learned from Scratch

The first few-shot learning experiment was conducted with two recurrent architectures, an LSTM and a GRU, both initialized with embeddings learned from scratch. The LSTM model comprises approximately 22.2M parameters, while the GRU model includes 21.2M parameters. Both were trained using the RMSprop optimizer with a learning rate of 2×10^{-5} and a dropout of 0.3. Tables 5.22 and 5.23 present the performance metrics for values of $K = 1$ to 10.

Table 5.22: Few-Shot Learning performance with LSTM on MSKCC

K	Test Acc.	F1	Bal. Acc.	Recall	Precision
1	0.5060	0.3315	0.4757	0.3961	0.2850
2	0.5241	0.3324	0.4852	0.3831	0.2935
3	0.5402	0.2541	0.4609	0.2532	0.2549
4	0.5843	0.4000	0.5467	0.4481	0.3613
5	0.4418	0.4184	0.4991	0.6494	0.3086
6	0.5321	0.4071	0.5286	0.5195	0.3347
7	0.5843	0.4266	0.5610	0.5000	0.3720
8	0.5964	0.4036	0.5536	0.4416	0.3716
9	0.4036	0.3800	0.4553	0.5909	0.2800
10	0.4277	0.3871	0.4710	0.5844	0.2894

Table 5.23: Few-Shot Learning performance with GRU on MSKCC

K	Test Acc.	F1	Bal. Acc.	Recall	Precision
1	0.5120	0.3520	0.4890	0.4286	0.2986
2	0.5080	0.3860	0.5058	0.5000	0.3143
3	0.4538	0.3761	0.4755	0.5325	0.2908
4	0.5361	0.3456	0.4975	0.3961	0.3065
5	0.4900	0.3553	0.4802	0.4545	0.2917
6	0.4618	0.3679	0.4742	0.5065	0.2889
7	0.4598	0.3487	0.4620	0.4675	0.2780
8	0.5040	0.3233	0.4706	0.3831	0.2917
9	0.4900	0.4381	0.5322	0.6429	0.3322
10	0.5723	0.4196	0.5523	0.5000	0.3615

The results of few-shot experiments with LSTM and GRU reveal considerable instability as the number of examples per class increases. Both models struggle to generalize with limited data, showing fluctuating accuracies and modest F1 scores that rarely exceed 0.45. LSTM tends to perform better at moderate values of K, while GRU shows improvements at higher K, yet neither architecture achieves stable or robust performance.

Experiment 2: Few-Shot Learning with LSTM and GRU using Pre-trained Word Embeddings (GloVe 300d and FastText 300d)

In this experiment, we evaluated bidirectional LSTM and GRU models using pre-trained word embeddings (GloVe 300d and FastText 300d). Each model has an embedding dimension of 300, 500 hidden units in the recurrent layer, and a dropout rate of 0.3. The bidirectional output is mapped through a fully connected layer to a single output unit for binary classification.

All models were trained using the RMSprop optimizer with a learning rate of 2×10^{-5}. The GRU models contain approximately 14.7M parameters, while the LSTM models have around 15.5M parameters.

Tables 5.24 and 5.25 summarize the performance metrics for few-shot settings with $K = 1$ to 10.

Table 5.24: Few-Shot Learning performance with Glove embeddings (300d)

K	GRU					LSTM				
	Acc	F1	BalAcc	Recall	Prec	Acc	F1	BalAcc	Recall	Prec
1	0.396	0.416	0.478	0.695	0.296	0.343	0.460	0.498	0.903	0.308
2	0.586	0.452	0.577	0.552	0.383	0.426	0.476	0.541	0.844	0.332
3	0.339	0.349	0.403	0.571	0.251	0.474	0.376	0.485	0.513	0.297
4	0.351	0.436	0.479	0.812	0.298	0.572	0.472	0.585	0.617	0.382
5	0.460	0.390	0.487	0.558	0.300	0.394	0.461	0.516	0.838	0.318
6	0.488	0.414	0.515	0.584	0.320	0.466	0.470	0.549	0.766	0.339
7	0.488	0.324	0.463	0.396	0.274	0.566	0.337	0.509	0.357	0.320
8	0.402	0.411	0.477	0.675	0.296	0.432	0.463	0.531	0.792	0.327
9	0.392	0.468	0.522	0.864	0.321	0.556	0.316	0.494	0.331	0.302
10	0.434	0.387	0.474	0.578	0.291	0.621	0.145	0.478	0.104	0.239

Table 5.25: Few-Shot Learning performance with FastText embeddings (300d)

K	LSTM					GRU				
	Acc	F1	BalAcc	Recall	Prec	Acc	F1	BalAcc	Recall	Prec
1	0.645	0.227	0.513	0.169	0.347	0.311	0.472	0.500	0.994	0.309
2	0.661	0.045	0.485	0.026	0.174	0.404	0.457	0.516	0.812	0.318
3	0.337	0.478	0.515	0.981	0.316	0.384	0.468	0.520	0.877	0.319
4	0.355	0.475	0.517	0.942	0.317	0.681	0.059	0.502	0.033	0.333
5	0.361	0.448	0.493	0.838	0.306	0.568	0.295	0.492	0.292	0.298
6	0.516	0.477	0.571	0.714	0.358	0.323	0.471	0.503	0.974	0.311
7	0.518	0.344	0.488	0.409	0.297	0.335	0.469	0.505	0.948	0.311
8	0.602	0.272	0.502	0.240	0.314	0.548	0.303	0.485	0.318	0.290
9	0.347	0.477	0.517	0.961	0.317	0.442	0.393	0.481	0.584	0.296
10	0.633	0.335	0.540	0.299	0.380	0.639	0.280	0.525	0.227	0.365

The results of few-shot experiments with LSTM and GRU using pre-trained word embeddings (GloVe 300d and FastText 300d) show patterns broadly similar to those observed in Experiment 1 with embeddings learned from scratch. Both experiments exhibit fluctuating performance across different values of K and limited generalization in low-resource settings. LSTM generally performs slightly better at moderate K, while GRU occasionally shows improvements at higher K.

Compared to models trained from scratch, the use of pre-trained embeddings provides marginal gains in certain cases, such as slightly higher accuracies or F1 scores for some values of K. However, these improvements are inconsistent, and neither architecture achieves stable or robust performance. This indicates that while pre-trained embeddings add richer semantic information, they do not fully overcome the challenges inherent to few-shot learning.

Experiment 3: Few-Shot Learning with LSTM and GRU using Pre-trained Word Embeddings (GloVe 300d and FastText 300d)

In this experiment, we evaluated bidirectional LSTM and GRU models with Bahdanau attention using pre-trained word embeddings (GloVe 300d and FastText 300d). Each model has an embedding dimension

of 300, 500 hidden units in the recurrent layer, and a dropout rate of 0.3. The bidirectional output is mapped through a fully connected layer to a single output unit for binary classification.

All models were trained using the RMSprop optimizer with a learning rate of 2×10^{-5}. The GRU models contain approximately 13.4M parameters, while the LSTM models have around 13.8M parameters.

Tables 5.26 and 5.27 summarize the performance metrics for few-shot settings with $K = 1$ to 10.

Table 5.26: Few-Shot Learning performance with GloVe embeddings (300d) – Bahdanau Attention

K	GRU					LSTM				
	Acc	F1	BalAcc	Recall	Prec	Acc	F1	BalAcc	Recall	Prec
1	0.311	0.473	0.502	1.000	0.310	0.359	0.476	0.521	0.942	0.319
2	0.584	0.343	0.520	0.351	0.335	0.638	0.231	0.511	0.175	0.338
3	0.460	0.443	0.525	0.695	0.325	0.690	0.000	0.500	0.000	0.000
4	0.376	0.438	0.489	0.786	0.303	0.668	0.057	0.493	0.033	0.238
5	0.687	0.049	0.504	0.026	0.400	0.309	0.472	0.500	1.000	0.309
6	0.617	0.509	0.624	0.643	0.421	0.309	0.472	0.500	1.000	0.309
7	0.691	0.000	0.500	0.000	0.000	0.690	0.000	0.500	0.000	0.000
8	0.669	0.057	0.493	0.033	0.238	0.335	0.479	0.515	0.987	0.316
9	0.691	0.000	0.500	0.000	0.000	0.309	0.472	0.500	1.000	0.309
10	0.309	0.472	0.500	1.000	0.309	0.343	0.465	0.501	0.928	0.314

Table 5.27: Few-Shot Learning performance with FastText embeddings (300d) – Bahdanau Attention

K	GRU					LSTM				
	Acc	F1	BalAcc	Recall	Prec	Acc	F1	BalAcc	Recall	Prec
1	0.641	0.238	0.514	0.182	0.346	0.359	0.476	0.521	0.942	0.319
2	0.612	0.363	0.542	0.357	0.369	0.643	0.272	0.526	0.214	0.371
3	0.691	0.000	0.500	0.000	0.000	0.693	0.013	0.503	0.007	1.000
4	0.309	0.472	0.500	1.000	0.309	0.540	0.423	0.542	0.546	0.346
5	0.351	0.470	0.511	0.929	0.314	0.638	0.231	0.511	0.175	0.338
6	0.309	0.472	0.500	1.000	0.309	0.327	0.466	0.504	0.935	0.311
7	0.691	0.000	0.500	0.000	0.000	0.309	0.472	0.500	1.000	0.309
8	0.335	0.479	0.515	0.987	0.316	0.343	0.465	0.500	0.929	0.310
9	0.309	0.472	0.500	1.000	0.309	0.309	0.472	0.500	1.000	0.309
10	0.343	0.465	0.500	0.929	0.310	0.344	0.465	0.500	0.928	0.310

The results indicate similar patterns to previous experiments: performance fluctuates across different K values, with LSTM performing slightly better at moderate K and GRU sometimes outperforming at higher K. Pre-trained embeddings improve some metrics marginally, but overall performance remains unstable in low-resource settings.

Experiment 4: Few-Shot Learning with Model-based LSTM and GRU with Bahdanau Attention using BioMedNLP Contextual Embeddings

In this experiment, we evaluated model-based LSTM and GRU architectures with Bahdanau attention using BioMedNLP contextual embeddings. The LSTM model has 2,580,865 parameters, while the GRU model has 1,976,833 parameters. Both models use a bidirectional recurrent layer with 286 hidden units, Bahdanau attention applied on the recurrent outputs, and a fully connected layer for binary classification. A dropout rate of 0.1 is applied during training.

Tables 5.28 and 5.29 summarize the few-shot performance for $K = 1$ to 10.

The results of few-shot experiments with LSTM and GRU using BioMedNLP contextual embeddings show that LSTM with attention tends to achieve higher accuracy in very low K settings, while GRU exhibits more variable performance across different K. Both architectures demonstrate limited stability in extremely low-resource scenarios, with some K values yielding zero F1 scores or unbalanced precision/recall. Compared to previous experiments using static embeddings, contextual embeddings provide richer semantic information but still do not fully address the challenges of few-shot learning in biomedical text.

Table 5.28: Few-Shot Learning performance with BioMedNLP embeddings using LSTM with Bahdanau Attention

K	Acc	F1	BalAcc	Recall	Prec
1	0.7249	0.3568	0.5929	0.2468	0.6441
2	0.6908	0.0000	0.5000	0.0000	0.0000
3	0.6968	0.0736	0.5151	0.0390	0.6667
4	0.6908	0.0000	0.5000	0.0000	0.0000
5	0.6988	0.0854	0.5184	0.0455	0.7000
6	0.6908	0.0000	0.5000	0.0000	0.0000
7	0.7149	0.3545	0.5874	0.2532	0.5909
8	0.7088	0.2786	0.5633	0.1818	0.5957
9	0.7068	0.1098	0.5278	0.0584	0.9000
10	0.7149	0.2680	0.5641	0.1688	0.6500

Table 5.29: Few-Shot Learning performance with BioMedNLP embeddings using GRU with Bahdanau Attention

K	Acc	F1	BalAcc	Recall	Prec
1	0.3092	0.4724	0.5000	1.0000	0.3092
2	0.5904	0.2917	0.5026	0.2727	0.3134
3	0.7309	0.4417	0.6241	0.3442	0.6163
4	0.5161	0.5051	0.5941	0.7987	0.3694
5	0.6948	0.4198	0.6015	0.3571	0.5093
6	0.6988	0.5833	0.6941	0.6818	0.5097
7	0.6867	0.2642	0.5473	0.1818	0.4828
8	0.6867	0.5806	0.6908	0.7013	0.4954
9	0.6908	0.0000	0.5000	0.0000	0.0000
10	0.6546	0.4724	0.6119	0.5000	0.4477

Figure 5.15 further illustrates how the balanced accuracy of the GRU with Bahdanau attention varies with K, highlighting peaks around $K = 3$ and $K = 6$.

5.2 Experiments on Medical Abstract Dataset

Following the experiments conducted on the MSKCC dataset, where we addressed a binary classification task, we now extend our evaluation to the Medical Abstract Dataset, which involves multi-class classification. This dataset contains biomedical text samples extracted from peer-reviewed publications. It was partitioned into separate training (70%), validation (15%), and test (15%) sets to ensure consistent evaluation across models. In total, the dataset consisted of 9,817 samples, with 8,085 samples used for training, 1,732 for validation, and 1,733 reserved for independent testing.

5.2.1 Experiment with Machine Learning Models — Linear SVM and Random Forest

In this first experiment on the Medical Abstract Dataset, we aimed to establish classical machine learning baselines before transitioning to deep learning approaches. As with the MSKCC dataset, the biomedical abstracts were represented using a `TF-IDF` vectorization scheme with a maximum of 5000 features and an n-gram range of (1, 2). The resulting sparse vectors served as input to both Linear SVM and Random Forest classifiers.

Experiment 1: Without applying SMOTE

We first trained the models on the imbalanced dataset without applying any resampling strategy. Table 5.30 details the pipeline configurations, while Table 5.31 reports the evaluation metrics obtained under this baseline condition.

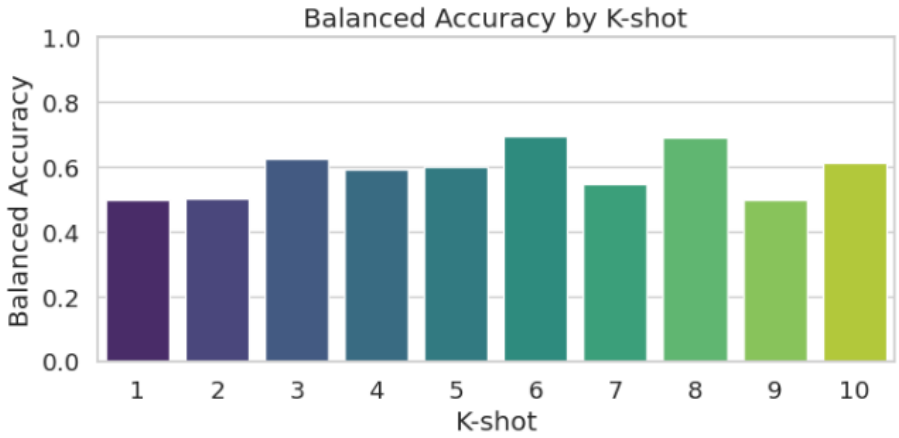


Figure 5.15: Histogram of balanced accuracy across different values of K for the GRU with Bahdanau attention using BioMedNLP contextual embeddings.

Table 5.30: Machine Learning Pipeline Configurations

Model	**Pipeline Components and Parameters**
Linear SVM	`TfidfVectorizer`: max_features=5000, ngram_range=(1, 2) `SVC`: C=0.1, kernel='linear', gamma='scale', class_weight='balanced', probability=True
Random Forest	`TfidfVectorizer`: max_features=5000, ngram_range=(1, 2) `RandomForestClassifier`: max_depth=10, n_estimators=200, min_samples_split=2, class_weight='balanced', random_state=42

The Linear SVM achieves stronger results than the Random Forest, particularly in balanced accuracy and AUC. However, both models exhibit relatively modest F1-scores due to the inherent class imbalance. The confusion matrices and ROC curves for both models are presented in Figures 5.16 to illustrate performance across different classes.

Experiment 2: With SMOTE applied

Next, we applied the Synthetic Minority Oversampling Technique (SMOTE) to rebalance the training set. Table 5.32 reports the evaluation metrics obtained after oversampling.
After applying SMOTE, the Linear SVM maintains stable performance with slight improvements in F1-score and AUC. By contrast, the Random Forest shows a decrease in balanced accuracy and AUC, suggesting oversampling may slightly destabilize the ensemble in this multi-class setting. Figures 5.17 illustrates these outcomes with the corresponding confusion matrices and ROC curves.

Conclusion

The relatively low balanced accuracy, F1-score, recall, and precision highlight the challenge of classifying the Medical Abstract Dataset with traditional machine learning models.

Linear SVM shows stable performance, with high AUC values indicating good discriminative ability despite modest F1-scores. Random Forest benefits somewhat from SMOTE in F1-score, but slight reductions in balanced accuracy and AUC suggest sensitivity to oversampling in multi-class settings.

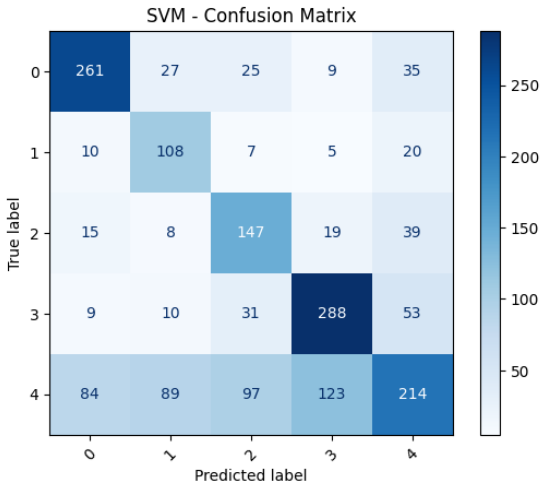
Figures 5.16 and 5.17 illustrate these results, showing that both class imbalance and dataset complexity pose challenges for classical classifiers, even when discrimination between classes (as measured by AUC) is strong.

Table 5.31: Performance Metrics for Machine Learning Models on Medical Abstract Dataset Without SMOTE

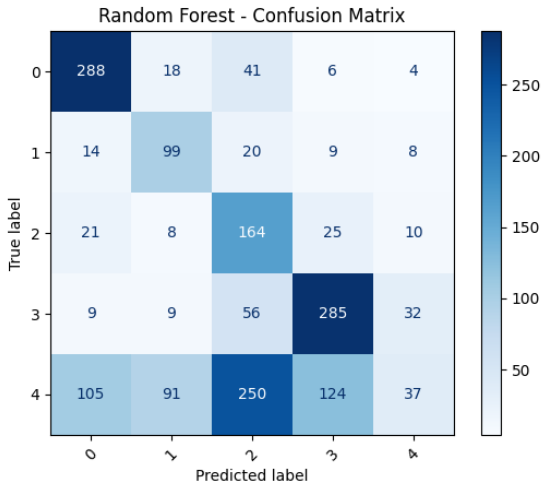
Model	Bal. Acc.	F1-score	Recall	Precision	AUC
Linear SVM	0.637	0.588	0.637	0.571	0.853
Random Forest	0.595	0.494	0.595	0.490	0.825

Table 5.32: Performance Metrics for Machine Learning Models on Medical Abstract Dataset With SMOTE

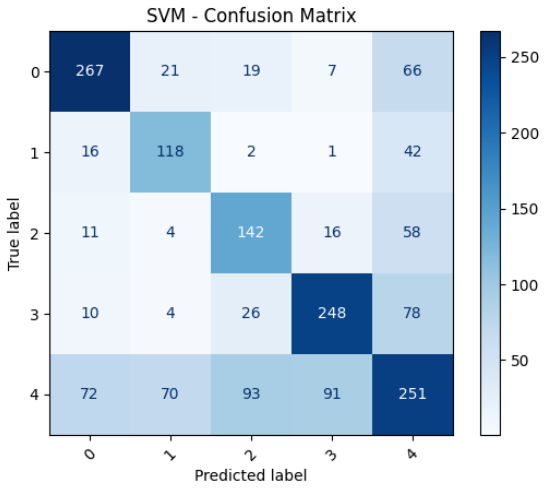
Model	Bal. Acc.	F1-score	Recall	Precision	AUC
Linear SVM	0.618	0.601	0.618	0.590	0.858
Random Forest	0.529	0.520	0.529	0.514	0.812



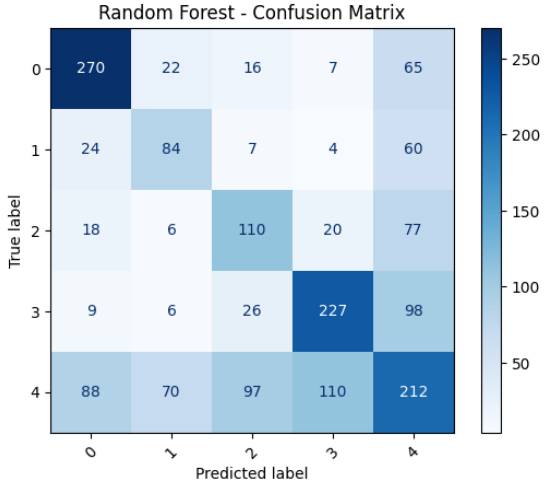
SVM Confusion Matrix (No SMOTE)



Random Forest Confusion Matrix (No SMOTE)

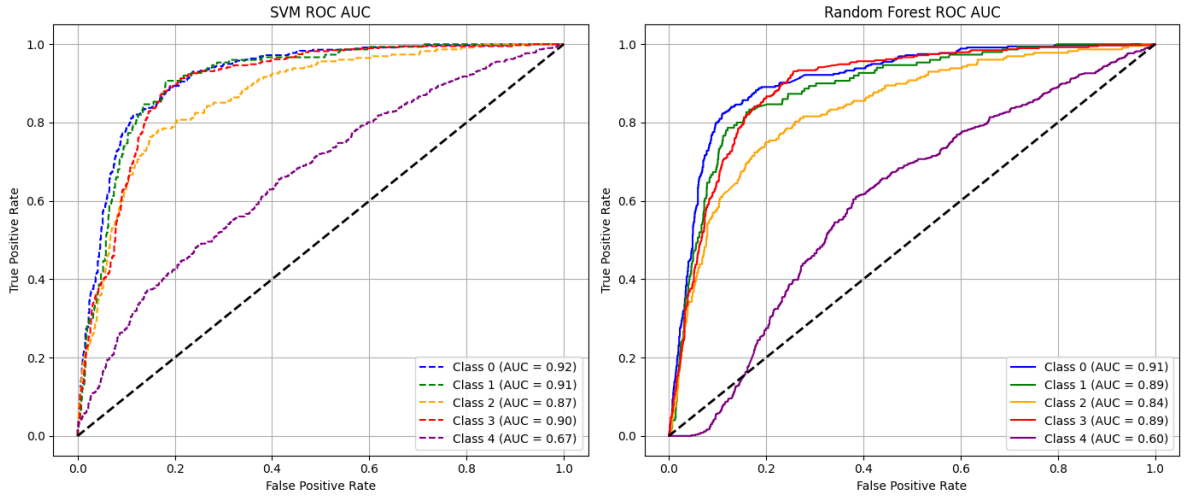


SVM Confusion Matrix (With SMOTE)

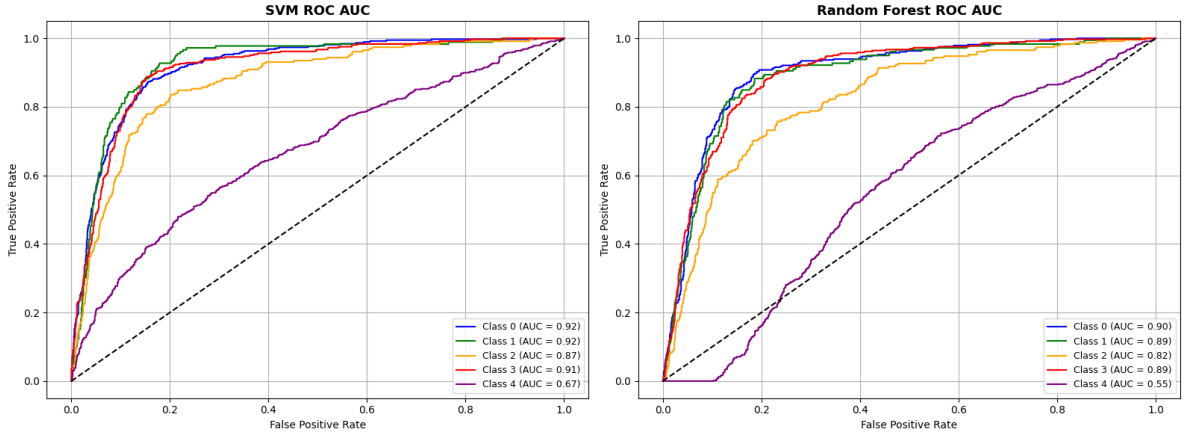


Random Forest Confusion Matrix (With SMOTE)

Figure 5.16: Confusion matrices for SVM and Random Forest on the Medical Abstract Dataset, without and with SMOTE.



ROC Curves for SVM and Random Forest (No SMOTE)



ROC Curves for SVM and Random Forest (With SMOTE)

Figure 5.17: ROC curves comparing SVM and Random Forest on the Medical Abstract Dataset, with and without SMOTE.

5.2.2 Experiment with Deep Learning Models

As done with the first dataset MSKCC, the vocabulary for this experiment was constructed by tokenizing all training texts and removing tokens with a frequency lower than 2, resulting in a vocabulary of *20,931* unique tokens. We begin by training LSTM- and GRU-based models from scratch, followed by models initialized with pre-trained word embeddings (GloVe (300d) and Fasttext (300d)). Subsequently, we incorporate an attention mechanism (Bahdanau attention) to enhance sequence modeling, and finally, we evaluate models using BioMedNLP contextual embeddings.

During model selection and optimization[2], we systematically explored key hyperparameters, including the choice of optimizer (Adam, RMSprop, and AdamW), batch size (8, 16, and 32), dropout rates, and hidden layer dimensions

Experiment 1: Model-based LSTM and GRU with embeddings learned from scratch

We start by evaluating LSTM and GRU models trained from scratch, without applying SMOTE or class weighting. The batch size was set to 16 and the model depth to 1. The following tables summarize model configurations, hyperparameters, and the final evaluation metrics.

[2] `https://github.com/etoMvom/biomedical_text_classification`

Experiment 1: Without SMOTE or Class Weighting

Table 5.33 shows the model configurations, while Table 5.34 reports the performance of LSTM and GRU models trained from scratch.

Table 5.33: Summary of model configurations, hyperparameters, and optimizers (batch size = 16, depth = 1)

Model	Embedding Dim.	Hidden Dim.	Dropout	Depth	Batch Size	Total Params	Optimizer & LR
LSTM	500	500	0.1	1	16	10,739,555	RMSprop, 1×10^{-3}
GRU	480	480	0.2	1	16	9,727,515	RMSprop, 1×10^{-4}

Table 5.34: Performance of LSTM and GRU models without SMOTE and Class Weighting

Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc	F1	Bal. Acc	Prec.	Recall
LSTM	26	0.92	1.55	22.6%	8.3%	20.0%	5.1%	22.6%
GRU	46	0.67	1.32	50.0%	45.7%	44.2%	42.4%	50.0%

Experiment 2: Applying SMOTE

Tables 5.35 and 5.36 summarize the configurations and performance with SMOTE applied (batch size = 16, depth = 1).

Table 5.35: Model configurations, hyperparameters, and optimizers with SMOTE applied

Model	Embedding Dim.	Hidden Dim.	Dropout	Depth	Batch Size	Total Params	Optimizer & LR
LSTM	450	500	0.2	1	16	13,232,855	RMSprop, 1×10^{-3}
GRU	350	480	0.2	1	16	9,727,515	RMSprop, 1×10^{-4}

Table 5.36: Performance of LSTM and GRU models with SMOTE

Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
LSTM	26	1.46	1.6634	8.66%	1.38%	20.00%	0.75%	8.66%
GRU	31	1.04	1.3202	41.89%	33.96%	45.38%	50.12%	41.89%

Experiment 3: Applying Class Weighting

Tables 5.37 and 5.38 summarize configurations and performance with class weighting (batch size = 16, depth = 1).

Table 5.37: Model configurations, hyperparameters, and optimizers with Class Weighting applied

Model	Embedding Dim.	Hidden Dim.	Dropout	Depth	Batch Size	Total Params	Optimizer & LR
LSTM	450	500	0.2	1	16	13,232,855	RMSprop, 1×10^{-3}
GRU	350	400	0.2	1	16	9,135,355	RMSprop, 1×10^{-4}

Table 5.38: Performance of LSTM and GRU models with Class Weighting

Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
LSTM	26	0.91	1.6793	8.66%	1.38%	20.00%	0.75%	8.66%
GRU	35	0.51	1.3681	46.86%	40.75%	53.07%	47.74%	46.86%

Conclusion

The experiments with LSTM and GRU models trained from scratch, using a batch size of 16 and a model depth of 1, highlight clear differences in performance between the two architectures. Overall, GRU consistently outperforms LSTM across baseline, SMOTE, and class weighting scenarios, achieving higher balanced accuracy and F1-scores, which suggests that GRU better captures sequential dependencies and handles class imbalance. Applying SMOTE provides modest gains, particularly in recall, but sometimes reduces precision, indicating that synthetic oversampling alone cannot fully compensate for imbalanced data in models trained from scratch. Incorporating class weighting benefits GRU more noticeably than LSTM, allowing it to better focus on minority classes, while LSTM shows minimal improvement. In terms of training efficiency, LSTM reaches early stopping in fewer epochs, but GRU, despite requiring longer per epoch, converges to higher-quality solutions. When compared with classical machine learning models such as Random Forest or linear SVM, deep sequence models trained from scratch struggle to consistently surpass their performance, particularly in efficiency and interpretability. Classical models can achieve competitive or higher balanced accuracy with proper feature engineering and class balancing, while GRU offers potential advantages when sequential patterns are strong. These findings suggest that future improvements could come from leveraging pre-trained embeddings.

5.2.2.1 Experiment with model-based LSTM and GRU using pre-trained embeddings (GloVe-300d and FastText-300d)

Tables 5.39 summarizes the model configurations, hyperparameters, and optimizers used for the LSTM and GRU models with pre-trained embeddings on Medical Abstract Dataset. The batch size was set to 16 and the depth was set to 1.These same configurations were consistently applied across all subsequent experiments, including those with SMOTE and class weighting.

Table 5.39: Model configurations, hyperparameters, and optimizers with pre-trained embeddings

Embedding	Model	Hidden Dim.	Dropout	Total Params	Optimizer & LR
GloVe-300d	LSTM	480	0.2	9,287,585	RMSprop, 1×10^{-3}
GloVe-300d	GRU	500	0.2	8,690,905	RMSprop, 1×10^{-3}
FastText-300d	LSTM	356	0.5	8,157,449	RMSprop, 1×10^{-3}
FastText-300d	GRU	500	0.3	8,690,905	RMSprop, 1×10^{-3}

Experiment 1: Without applying SMOTE or Class Weighting

Table 5.40: Performance of LSTM and GRU models without SMOTE or class weighting

Embedding	Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
GloVe-300d	LSTM	13	0.78	1.5209	35.03%	18.17%	20.00%	12.27%	35.03%
GloVe-300d	GRU	13	0.68	1.5100	35.03%	18.17%	20.00%	12.27%	35.03%
FastText-300d	LSTM	16	0.52	1.5119	35.03%	18.17%	20.00%	12.27%	35.03%
FastText-300d	GRU	13	0.67	1.5171	35.03%	18.17%	20.00%	12.27%	35.03%

Experiment 2: With SMOTE applied

Table 5.41: Performance of LSTM and GRU models with SMOTE applied

Embedding	Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
GloVe-300d	LSTM	13	0.84	1.6173	13.16%	3.06%	20.00%	1.73%	13.16%
GloVe-300d	GRU	13	1.07	1.6091	20.60%	7.04%	20.00%	4.24%	20.60%
FastText-300d	LSTM	13	0.84	1.6166	8.66%	1.38%	20.00%	0.75%	8.66%
FastText-300d	GRU	13	1.07	1.6023	35.03%	18.17%	20.00%	12.27%	35.03%

Experiment 3: With Class Weighting applied

Table 5.42: Performance of LSTM and GRU models with class weighting

Embedding	Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
GloVe-300d	LSTM	13	0.84	1.6173	13.16%	3.06%	20.00%	1.73%	13.16%
GloVe-300d	GRU	13	1.07	1.6091	20.60%	7.04%	20.00%	4.24%	20.60%
FastText-300d	LSTM	13	0.84	1.6166	8.66%	1.38%	20.00%	0.75%	8.66%
FastText-300d	GRU	13	1.07	1.6023	35.03%	18.17%	20.00%	12.27%	35.03%

Conclusion

GRU models consistently outperformed LSTM models in terms of test accuracy and balanced accuracy. However, due to the extreme class imbalance in Medical Abstract Dataset, SMOTE and class weighting had minimal impact on overall model performance, indicating that additional techniques may be required to handle the dataset effectively.

5.2.2.2 Experiment with LSTM and GRU with Bahdanau Attention and word embeddings

Adding Bahdanau attention aims to enhance the models' ability to focus on important words in sequences. We use the same pre-trained embeddings and repeat the imbalance handling experiments.

Experiment 1: Without SMOTE or Class Weighting

Table 5.43: Performance of LSTM and GRU with Bahdanau Attention (No SMOTE / Class Weighting)

Model	Embedding	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
LSTM	GloVe (300d)	14	0.88	0.8632	62.78%	61.96%	62.28%	62.30%	62.78%
GRU	GloVe (300d)	16	0.69	0.8851	62.61%	61.36%	64.31%	62.37%	62.61%
LSTM	FastText (300d)	14	0.88	0.8927	63.42%	62.36%	63.45%	62.70%	63.42%
GRU	FastText (300d)	18	0.70	0.9870	59.20%	57.58%	63.76%	59.52%	59.20%

Experiment 2: With SMOTE applied

Table 5.44: Performance of LSTM and GRU with Bahdanau Attention and SMOTE

Model	Embedding	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
LSTM	GloVe (300d)	15	0.64	0.8725	62.67%	62.22%	63.79%	62.32%	62.67%
GRU	GloVe (300d)	18	0.82	0.9058	60.88%	59.28%	63.91%	61.30%	60.88%
LSTM	FastText (300d)	14	0.94	0.9880	60.47%	60.41%	64.42%	62.05%	60.47%
GRU	FastText (300d)	18	0.82	1.0744	54.59%	53.80%	58.18%	55.00%	54.59%

Experiment 3: With Class Weighting applied

Table 5.45: Performance of LSTM and GRU with Bahdanau Attention and Class Weighting

Model	Embedding	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Prec.	Recall
LSTM	GloVe (300d)	6	0.89	0.8207	63.42%	61.87%	68.57%	63.55%	63.42%
GRU	GloVe (300d)	8	0.70	0.8612	60.07%	56.93%	68.40%	63.66%	60.07%
LSTM	FastText (300d)	6	0.90	0.9025	63.47%	60.97%	68.43%	64.78%	63.47%
GRU	FastText (300d)	9	0.70	1.0300	54.88%	50.06%	63.77%	57.99%	54.88%

Conclusion

LSTM models consistently outperform GRU models across the different balancing strategies. When no balancing is applied, LSTM with FastText achieves the best accuracy (63.42%) and F1 (62.36%), slightly higher than GloVe. With SMOTE, performance remains comparable to the unbalanced setting for LSTM

(62.67% with GloVe, 60.47% with FastText), while GRU degrades more notably, especially with FastText (54.59% accuracy, 53.80% F1). By contrast, applying class weighting provides the strongest improvements in balanced accuracy and recall, with LSTM + GloVe reaching 68.57% balanced accuracy and LSTM + FastText 68.43%. This demonstrates that class weighting is more effective than oversampling for handling imbalance. Incorporating Bahdanau attention preserves or slightly improves the models' ability to focus on informative words, contributing to more stable and reliable classification performance.

5.2.2.3 Experiment with Model-based LSTM and GRU with Bahdanau Attention using BioMedNLP Contextual Embeddings

We evaluate LSTM and GRU models with Bahdanau attention using BioMedNLP contextual embeddings. The experiments explore three handling strategies: no SMOTE or class weighting, SMOTE, and class weighting. All models are trained with Adam optimizer and learning rate 1×10^{-4} unless otherwise stated.

Experiment 1: Without SMOTE or Class Weighting

Table 5.46: Model configurations for LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (without SMOTE and class weighting)

Model	Hidden Dim.	Dropout	Attention Dim.	Total Params	Optimizer & LR
LSTM	286	0.4	286	2,583,157	Adam, 1×10^{-4}
GRU	286	0.4	286	2,583,157	Adam, 1×10^{-4}

Table 5.47: Performance of LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (without SMOTE and class weighting)

Model	Epoch (ES)	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Precision	Recall
GRU	12	6.09	0.8535	64.28%	63.42%	65.94%	63.82%	64.28%
LSTM	7	7.26	0.8727	62.67%	60.86%	67.49%	63.01%	62.67%

Experiment 2: With SMOTE

Table 5.48: Model configurations for LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (with SMOTE)

Model	Hidden Dim.	Dropout	Attention Dim.	Total Params	Optimizer & LR
LSTM	286	0.4	286	2,583,157	Adam, 1×10^{-4}
GRU	156	0.5	156	917,285	Adam, 1×10^{-4}

Table 5.49: Performance of LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (with SMOTE)

Model	Epoch	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Precision	Recall
GRU	17	6.39	0.8455	65.48%	63.93%	65.48%	64.03%	64.51%
LSTM	9	7.26	0.8796	62.90%	61.05%	67.20%	63.14%	62.90%

Experiment 3: With Class Weighting

Table 5.50: Model configurations for LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (with class weighting applied)

Model	Hidden Dim.	Dropout	Attention Dim.	Total Params	Optimizer & LR
LSTM	286	0.4	286	2,583,157	Adam, 1×10^{-4}
GRU	286	0.4	286	2,583,157	Adam, 1×10^{-4}

Table 5.51: Performance of LSTM and GRU with Bahdanau attention using BioMedNLP embeddings (with class weighting)

Model	Epoch	Time (min/epoch)	Test Loss	Test Acc.	F1	Bal. Acc.	Precision	Recall
GRU	15	5.49	0.8664	63.53%	61.39%	70.04%	65.42%	63.53%
LSTM	9	6.12	0.8422	61.51%	58.96%	69.92%	65.39%	61.51%

Conclusion

GRU models generally perform better than LSTM models in terms of test accuracy and F1 score. Without balancing strategies, GRU reached higher accuracy (64.28% vs. 62.67%) and F1 score (63.42% vs. 60.86%). With SMOTE, GRU again achieved better results with accuracy of 65.48% and F1 score of 63.93%, while LSTM remained lower. When class weighting was applied, both models obtained their highest balanced accuracy (GRU: 70.04%, LSTM: 69.92%), but GRU continued to provide higher accuracy (63.53% vs. 61.51%) and F1 score (61.39% vs. 58.96%). This shows that class weighting is the most effective strategy for handling imbalance, particularly for GRU.

Training efficiency also differed: GRU required less time per epoch (5.49–6.39 min) compared with LSTM (6.12–7.26 min). The best-performing configuration was GRU with Bahdanau attention and class weighting using BioMedNLP embeddings, as it offered the most favorable balance of accuracy, F1 score, robustness against imbalance, and runtime.

Figures 5.18–5.24 depict model loss, balanced accuracy, confusion matrices, and ROC curves for GRU and LSTM with BioMedNLP embeddings, highlighting the performance gap and the benefit of class weighting for imbalance; combined with the runtime statistics, they also make clear that these models require more training time per epoch.

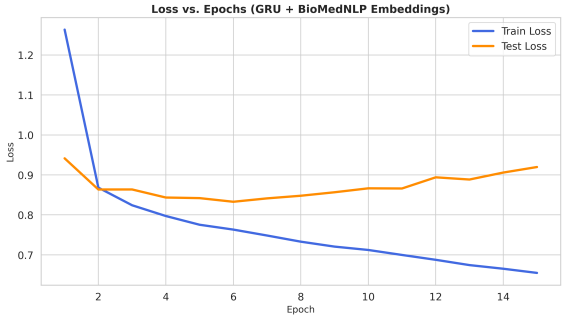


Figure 5.18: GRU + Attention (BioMedNLP) - Loss

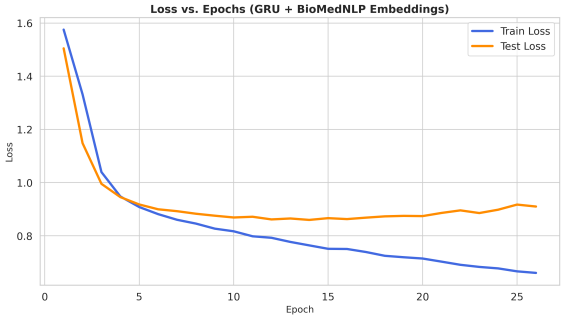


Figure 5.19: LSTM + Attention (BioMedNLP) - Loss

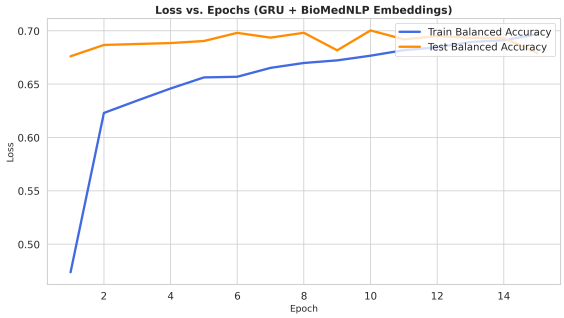


Figure 5.20: GRU + Attention (BioMedNLP) - Balanced Accuracy

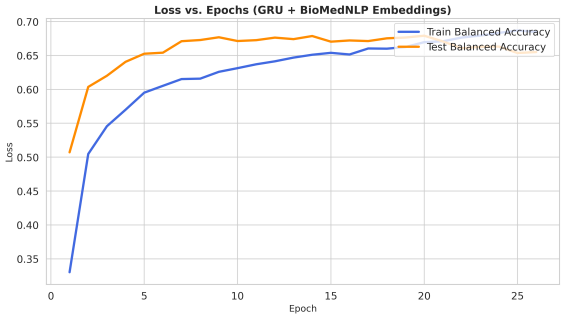


Figure 5.21: LSTM + Attention (BioMedNLP) - Balanced Accuracy

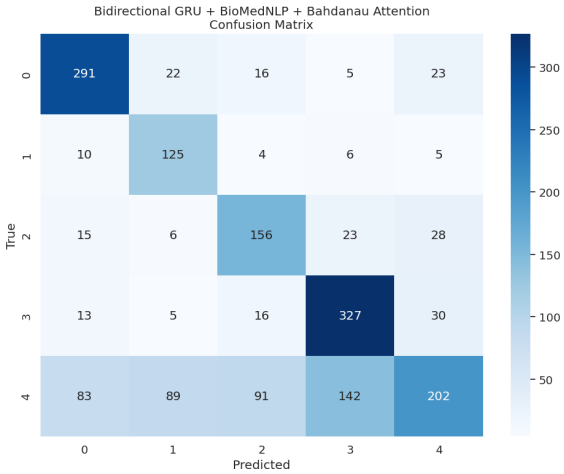


Figure 5.22: GRU + Attention (BioMedNLP) - Confusion Matrix

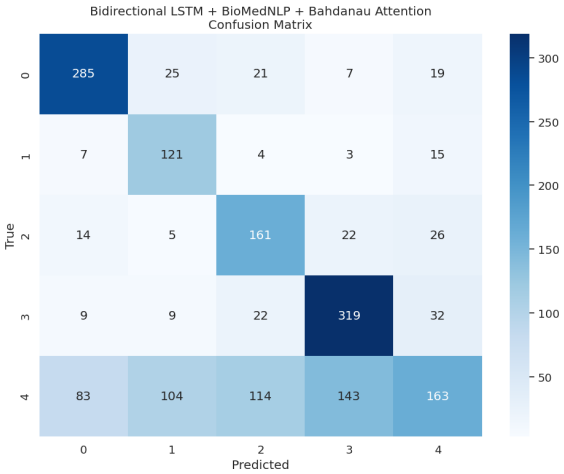


Figure 5.23: LSTM + Attention (BioMedNLP) - Confusion Matrix

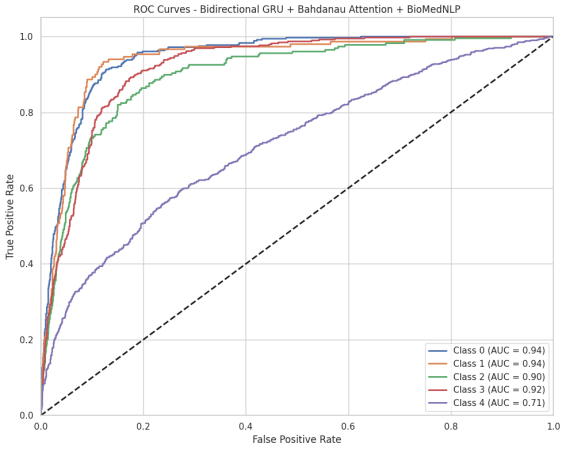


Figure 5.24: GRU + Attention (BioMedNLP) - ROC

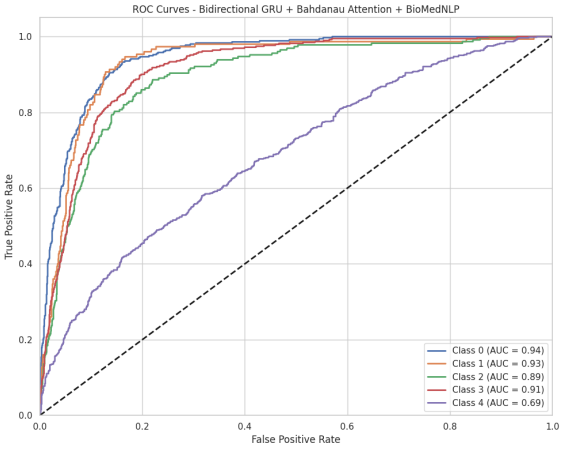


Figure 5.25: LSTM + Attention (BioMedNLP) - ROC

5.2.2.4 Few-Shot Learning on the Medical Abstract Dataset

As in the MSKCC experiments, only models that passed validation and were optimized during model selection were retained for evaluation, ensuring fair and reliable baselines. For each run, the K abstracts are randomly drawn from the training set, which, given their variable token lengths described earlier, naturally introduces additional variability in the reported metrics.

Experiment 1: Few-Shot Learning with LSTM and GRU using Embeddings Learned from Scratch

The first few-shot learning experiment was conducted using two recurrent architectures, LSTM and GRU, both initialized with embeddings learned from scratch. The LSTM model has approximately 10.2M parameters, whereas the GRU model has 9.5M parameters. Both models were trained with the RMSprop optimizer at a learning rate of 2×10^{-5} and a dropout rate of 0.4. Tables 5.52 and 5.53 summarize the performance metrics for $K = 1$ to 10.

Table 5.52: Few-Shot Learning performance with LSTM on the medical abstract dataset

K	Test Acc.	F1	Bal. Acc.	Recall	Precision
1	0.3503	0.1817	0.2000	0.3503	0.1227
2	0.1316	0.0306	0.2000	0.1316	0.0173
3	0.1316	0.0306	0.2000	0.1316	0.0173
4	0.0866	0.0138	0.2000	0.0866	0.0075
5	0.1316	0.0306	0.2000	0.1316	0.0173
6	0.2256	0.0831	0.2000	0.2256	0.0509
7	0.3503	0.1817	0.2000	0.3503	0.1227
8	0.0866	0.0138	0.2000	0.0866	0.0075
9	0.3503	0.1817	0.2000	0.3503	0.1227
10	0.1316	0.0306	0.2000	0.1316	0.0173

Table 5.53: Few-Shot Learning performance with GRU on the medical abstract dataset

K	Test Acc.	F1	Bal. Acc.	Recall	Precision
1	0.0866	0.0138	0.2000	0.0866	0.0075
2	0.1316	0.0306	0.2000	0.1316	0.0173
3	0.0866	0.0138	0.2000	0.0866	0.0075
4	0.0866	0.0138	0.2000	0.0866	0.0075
5	0.3503	0.1817	0.2000	0.3503	0.1227
6	0.0866	0.0138	0.2000	0.0866	0.0075
7	0.2060	0.0704	0.2000	0.2060	0.0424
8	0.0866	0.0138	0.2000	0.0866	0.0075
9	0.1316	0.0306	0.2000	0.1316	0.0173
10	0.3503	0.1817	0.2000	0.3503	0.1227

The results indicate significant instability in few-shot learning performance for both models. LSTM shows occasional strong performance at K values like 1, 7, and 9, while GRU achieves its best results at K = 5 and 10. Overall, both architectures struggle to generalize with very few examples, evidenced by low F1 scores and balanced accuracy consistently around 0.2. GRU demonstrates slightly higher test accuracy at selective K values, but neither model attains stable or robust performance in this low-data regime.

Experiment 2: Few-Shot Learning with LSTM and GRU using Pre-trained Word Embeddings (GloVe 300d and FastText 300d)

In this experiment, we evaluated bidirectional LSTM and GRU models using pre-trained word embeddings (GloVe 300d and FastText 300d). Each model has an embedding dimension of 300, 450 hidden units

in the recurrent layer, and a dropout rate of 0.5. The bidirectional output is mapped through a fully connected layer to classify abstracts into 5 categories.

All models were trained using the RMSprop optimizer with a learning rate of 2×10^{-5}. The GRU models contain approximately 8.3M parameters, while the LSTM models have around 8.9M parameters.

Tables 5.54 and 5.55 summarize the performance metrics for few-shot settings with $K = 1$ to 10.

Table 5.54: Few-Shot Learning performance with GloVe embeddings (300d)

K	GRU					LSTM				
	Acc	F1	BalAcc	Recall	Prec	Acc	F1	BalAcc	Recall	Prec
1	0.0866	0.0138	0.2000	0.0866	0.0075	0.1316	0.0306	0.2000	0.1316	0.0173
2	0.0866	0.0138	0.2000	0.0866	0.0075	0.2060	0.0704	0.2000	0.2060	0.0424
3	0.1316	0.0306	0.2000	0.1316	0.0173	0.3503	0.1817	0.2000	0.3503	0.1227
4	0.1316	0.0306	0.2000	0.1316	0.0173	0.3503	0.1817	0.2000	0.3503	0.1227
5	0.0866	0.0138	0.2000	0.0866	0.0075	0.1316	0.0306	0.2000	0.1316	0.0173
6	0.1316	0.0306	0.2000	0.1316	0.0173	0.3503	0.1817	0.2000	0.3503	0.1227
7	0.2060	0.0704	0.2000	0.2060	0.0424	0.2256	0.0831	0.2000	0.2256	0.0509
8	0.2060	0.0704	0.2000	0.2060	0.0424	0.0866	0.0138	0.2000	0.0866	0.0075
9	0.0866	0.0138	0.2000	0.0866	0.0075	0.1316	0.0306	0.2000	0.1316	0.0173
10	0.3503	0.1817	0.2000	0.3503	0.1227	0.2060	0.0704	0.2000	0.2060	0.0424

Table 5.55: Few-Shot Learning performance with FastText embeddings (300d)

K	LSTM					GRU				
	Acc	F1	BalAcc	Recall	Prec	Acc	F1	BalAcc	Recall	Prec
1	0.1316	0.0306	0.2000	0.1316	0.0173	0.0866	0.0138	0.2000	0.0866	0.0075
2	0.2060	0.0704	0.2000	0.2060	0.0424	0.0866	0.0138	0.2000	0.0866	0.0075
3	0.3503	0.1817	0.2000	0.3503	0.1227	0.1316	0.0306	0.2000	0.1316	0.0173
4	0.3503	0.1817	0.2000	0.3503	0.1227	0.2060	0.0704	0.2000	0.2060	0.0424
5	0.1316	0.0306	0.2000	0.1316	0.0173	0.0866	0.0138	0.2000	0.0866	0.0075
6	0.3503	0.1817	0.2000	0.3503	0.1227	0.1316	0.0306	0.2000	0.1316	0.0173
7	0.2256	0.0831	0.2000	0.2256	0.0509	0.2060	0.0704	0.2000	0.2060	0.0424
8	0.0866	0.0138	0.2000	0.0866	0.0075	0.3503	0.1817	0.2000	0.3503	0.1227
9	0.1316	0.0306	0.2000	0.1316	0.0173	0.3503	0.1817	0.2000	0.3503	0.1227
10	0.2060	0.0704	0.2000	0.2060	0.0424	0.1316	0.0306	0.2000	0.1316	0.0173

The results of few-shot experiments with LSTM and GRU using pre-trained word embeddings show patterns broadly similar to models trained from scratch. Both architectures exhibit fluctuating performance across different values of K and limited generalization in low-resource settings. LSTM generally performs slightly better at moderate K, while GRU occasionally shows improvements at higher K.

Compared to models trained from scratch, pre-trained embeddings provide marginal gains in certain cases, such as slightly higher accuracies or F1 scores. However, these improvements are inconsistent, and neither architecture achieves stable performance, indicating that pre-trained embeddings alone do not fully overcome the challenges inherent to few-shot learning.

Experiment 3: Few-Shot Learning with LSTM and GRU with Bahdanau attention using Pre-trained Word Embeddings (GloVe 300d and FastText 300d)

In this experiment, we evaluate bidirectional LSTM and GRU models with Bahdanau attention, using pre-trained embeddings (FastText or GloVe, 300d). All models share a comparable architecture with ∼7.4–7.8M parameters, hidden size of 286, and dropout between 0.2 and 0.6. The bidirectional outputs are passed through a fully connected layer for multiclass prediction over the five abstract categories.

All models were trained using the RMSprop optimizer with a learning rate of 2×10^{-5}. The GRU models contain approximately 13.4M parameters, while the LSTM models have around 13.8M parameters.

Tables 5.56 and 5.57 summarize the performance metrics for few-shot settings with $K = 1$ to 10.

Table 5.56: Few-Shot Learning performance with GloVe embeddings (300d) – Bahdanau Attention

K	GRU					LSTM				
	Acc	F1	BalAcc	Recall	Prec	Acc	F1	BalAcc	Recall	Prec
1	0.311	0.473	0.502	1.000	0.310	0.359	0.476	0.521	0.942	0.319
2	0.584	0.343	0.520	0.351	0.335	0.638	0.231	0.511	0.175	0.338
3	0.460	0.443	0.525	0.695	0.325	0.690	0.000	0.500	0.000	0.000
4	0.376	0.438	0.489	0.786	0.303	0.668	0.057	0.493	0.033	0.238
5	0.687	0.049	0.504	0.026	0.400	0.309	0.472	0.500	1.000	0.309
6	0.617	0.509	0.624	0.643	0.421	0.309	0.472	0.500	1.000	0.309
7	0.691	0.000	0.500	0.000	0.000	0.690	0.000	0.500	0.000	0.000
8	0.669	0.057	0.493	0.033	0.238	0.335	0.479	0.515	0.987	0.316
9	0.691	0.000	0.500	0.000	0.000	0.309	0.472	0.500	1.000	0.309
10	0.309	0.472	0.500	1.000	0.309	0.343	0.465	0.501	0.928	0.314

Table 5.57: Few-Shot Learning performance with FastText embeddings (300d) – Bahdanau Attention

K	GRU					LSTM				
	Acc	F1	BalAcc	Recall	Prec	Acc	F1	BalAcc	Recall	Prec
1	0.641	0.238	0.514	0.182	0.346	0.359	0.476	0.521	0.942	0.319
2	0.612	0.363	0.542	0.357	0.369	0.643	0.272	0.526	0.214	0.371
3	0.691	0.000	0.500	0.000	0.000	0.693	0.013	0.503	0.007	1.000
4	0.309	0.472	0.500	1.000	0.309	0.540	0.423	0.542	0.546	0.346
5	0.351	0.470	0.511	0.929	0.314	0.638	0.231	0.511	0.175	0.338
6	0.309	0.472	0.500	1.000	0.309	0.327	0.466	0.504	0.935	0.311
7	0.691	0.000	0.500	0.000	0.000	0.309	0.472	0.500	1.000	0.309
8	0.335	0.479	0.515	0.987	0.316	0.343	0.465	0.500	0.929	0.310
9	0.309	0.472	0.500	1.000	0.309	0.309	0.472	0.500	1.000	0.309
10	0.343	0.465	0.500	0.929	0.310	0.344	0.465	0.500	0.928	0.310

The results indicate similar patterns to previous experiments: performance fluctuates across different K values, with LSTM performing slightly better at moderate K and GRU sometimes outperforming at higher K. Pre-trained embeddings improve some metrics marginally, but overall performance remains unstable in low-resource settings.

Experiment 4: Few-Shot Learning with Model-based LSTM and GRU with Bahdanau Attention using BioMedNLP Contextual Embeddings

In this experiment, we evaluate bidirectional LSTM and GRU models with Bahdanau attention, using BioMedNLP contextual embeddings (768d). The GRU model has 917,285 parameters, and the LSTM model has 1,206,197 parameters. Both architectures use a hidden size of 156, bidirectional recurrent layers, Bahdanau attention, and dropout of 0.4.

All models were trained using the RMSprop optimizer with a learning rate of 8×10^{-5}.

Tables 5.58 and 5.59 summarize the performance metrics for few-shot settings with $K = 1$ to 10.

Table 5.58: Few-Shot Learning performance with BioMedNLP embeddings – GRU + Bahdanau Attention

K	Acc	F1	BalAcc	Recall	Prec
1	0.2752	0.2086	0.3045	0.2752	0.3531
2	0.3705	0.3721	0.3561	0.3705	0.3881
3	0.4016	0.4066	0.4105	0.4016	0.4241
4	0.3658	0.3512	0.3993	0.3658	0.3911
5	0.3803	0.3720	0.4258	0.3803	0.4301
6	0.3474	0.3478	0.4092	0.3474	0.4353
7	0.2845	0.2369	0.3933	0.2845	0.3658
8	0.3566	0.3404	0.4392	0.3566	0.4090
9	0.4132	0.4118	0.4189	0.4132	0.4176
10	0.4357	0.3964	0.4789	0.4357	0.4364

Table 5.59: Few-Shot Learning performance with BioMedNLP embeddings – LSTM + Bahdanau Attention

K	Acc	F1	BalAcc	Recall	Prec
1	0.2314	0.2331	0.2598	0.2314	0.2743
2	0.3820	0.3774	0.3936	0.3820	0.4006
3	0.3162	0.3025	0.3577	0.3162	0.4074
4	0.3907	0.3736	0.4486	0.3907	0.4424
5	0.3814	0.3837	0.4101	0.3814	0.4069
6	0.3976	0.3648	0.4187	0.3976	0.4149
7	0.3618	0.3607	0.4105	0.3618	0.4186
8	0.3249	0.3085	0.4214	0.3249	0.3942
9	0.3595	0.3214	0.4562	0.3595	0.4217
10	0.4143	0.4072	0.4466	0.4143	0.4230

The results obtained from the few-shot learning experiments on the Medical Abstract Dataset show that performance fluctuates across different K values. The GRU model tends to perform better at higher K values, while LSTM occasionally outperforms at moderate K. Although BioMedNLP embeddings improve contextual representation, few-shot learning remains challenging in low-resource scenarios, with both models showing instability and limited generalization when trained on very few examples.

As shown in Figure 5.26, the evolution of the **balanced accuracy** for the GRU model using BioMedNLP embeddings is depicted across few-shot settings ($K = 1$–10). The plot highlights fluctuations in performance at lower K values and a general improvement as more support examples are provided. This visual summary complements the numeric results presented in Table 5.58, confirming that while GRU shows promising trends, few-shot learning on this dataset still requires more robust strategies to achieve consistent performance.

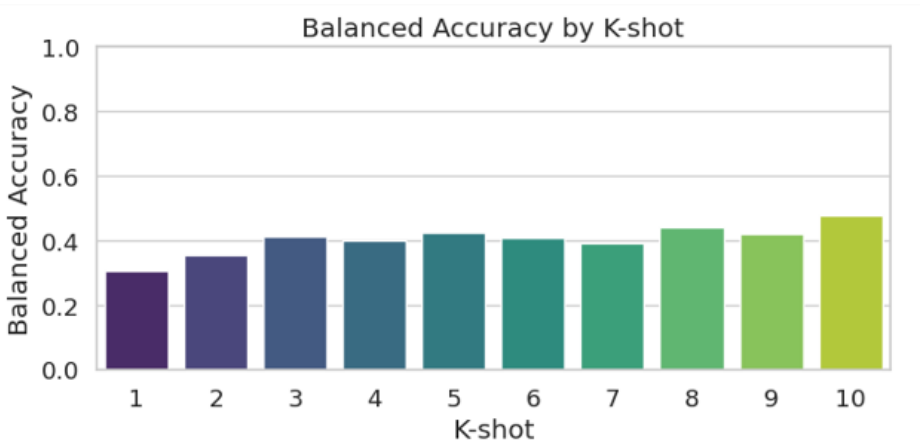


Figure 5.26: Evolution of balanced accuracy for the GRU model using BioMedNLP embeddings across different few-shot settings ($K = 1$–10).

Chapter 6

Conclusion

This thesis explored biomedical text classification using deep learning models, while employing machine learning models as baselines for comparison. The study focused on two datasets: the MSKCC dataset for binary classification of genetic mutations, and the Medical Abstracts dataset for multi-class disease categorization.

Machine learning models, including Linear SVM and Random Forest, served as reference points. On the MSKCC dataset, Random Forest achieved strong discriminative performance, especially when SMOTE was applied, improving recall and F1-score. Linear SVM showed consistent behavior across imbalance handling strategies. On the Medical Abstracts dataset, both models struggled with class imbalance, although SVM maintained relatively high AUC scores, demonstrating its robustness in certain scenarios.

Deep learning models, specifically GRU and LSTM architectures enhanced with Bahdanau attention, consistently outperformed traditional approaches. GRU models demonstrated faster training times and higher balanced accuracy and F1-scores compared to LSTM. The integration of BioMedNLP contextual embeddings significantly boosted performance, especially when combined with attention mechanisms, though it introduced greater computational demands.

To address class imbalance, both SMOTE and class weighting were evaluated. While SMOTE improved recall in some cases, class weighting proved to be more effective and stable, particularly for GRU-based models, leading to better overall performance across metrics.

Few-shot learning remained a challenging aspect, especially in low-resource disease categories. Models using BioMedNLP embeddings with Bahdanau attention showed promising generalization when combined with class weighting, but performance varied depending on the number of support examples (K), indicating the need for more adaptive few-shot strategies.

In conclusion, this research provides a comprehensive evaluation of model architectures, embedding techniques, and imbalance handling strategies for biomedical text classification. The findings emphasize the importance of combining contextual representations, attention mechanisms, and robust training strategies to build efficient and equitable NLP systems tailored to biomedical applications.

Chapter 7

Perspectives: BioWordVec, BioBERT, Prompt Engineering, Clinical Applications

Several future research directions emerge to enhance and extend this work:

The use of domain-specific embeddings such as BioWordVec, pretrained on large biomedical corpora, represents a promising path for enriching semantic representations. These word vectors, designed to capture domain-specific semantic relationships, offer a potential balance between traditional static embeddings and heavier contextual models. This approach could deliver strong performance while remaining computationally efficient, particularly in resource-constrained environments, without compromising the understanding of specialized biomedical language.

The rise of large generative language models, such as GPT or T5, opens new avenues for biomedical text classification through prompt engineering. Instead of retraining complex models, these architectures can be rapidly and efficiently adapted by formulating specific instructions (prompts). This technique holds promise for significantly reducing the need for annotated data and computational resources, while enabling flexible and customizable classification aligned with clinical or research needs.

The development of biomedical text classification systems must be accompanied by rigorous validation in collaboration with healthcare professionals. Such partnerships will help assess the relevance and reliability of models in real-world scenarios and ensure the integration of critical criteria such as interpretability and transparency. Tailoring models to the specific requirements of clinical practice—especially for decision-making and medical surveillance—will promote their adoption and impact within the healthcare domain.

To address the scarcity of annotated biomedical data, exploring semi-supervised learning and multitask transfer learning is an important direction. By leveraging unlabeled data and transferring knowledge across related tasks, these methods can improve model robustness and generalization, particularly for rare or emerging disease categories. This helps reduce dependency on costly annotated datasets while expanding the model's applicability.

To enable deployment in resource-constrained environments—such as embedded devices or healthcare facilities with limited infrastructure—techniques such as model compression, quantization, and distillation are essential. These approaches reduce model size and energy consumption with minimal impact on performance, facilitating wide-scale and practical deployment of biomedical NLP systems.

Finally, detecting, analyzing, and mitigating biases in both data and models remains a priority. Ensuring fairness and reliability in classification is critical, especially in the biomedical domain, where decisions can have direct consequences on patient health. Future work must follow a rigorous ethical framework to ensure transparency, accountability, and compliance with relevant standards, thereby fostering trust in automated systems.

This work thus lays the foundation for biomedical NLP systems that are powerful, practical, and ethically responsible—capable of supporting researchers and clinicians in the management and analysis of biomedical literature.

Appendices

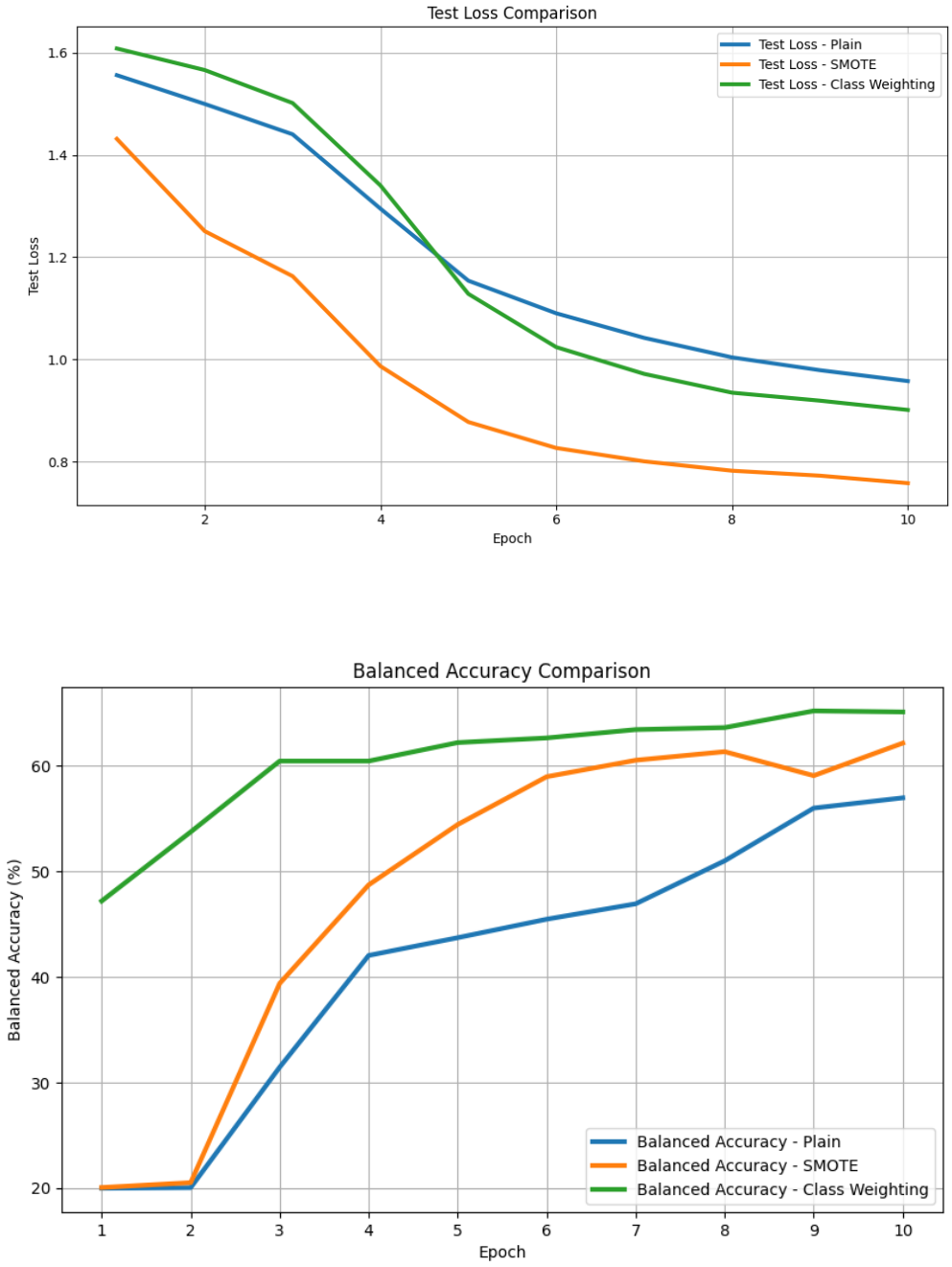


Figure 7.1: Bidirectional LSTM with BioMedNLP embeddings: Test Loss and Balanced Accuracy over epochs for different strategies (no resampling, class weighting, and SMOTE). The experiments on the medical abstracts dataset were conducted with the following hyperparameters: batch size of 64, BERT embedding dimension of 768, hidden dimension of 256, dropout rate of 0.7, and a learning rate of 5×10^{-5}. The models were optimized using the Adam optimizer.

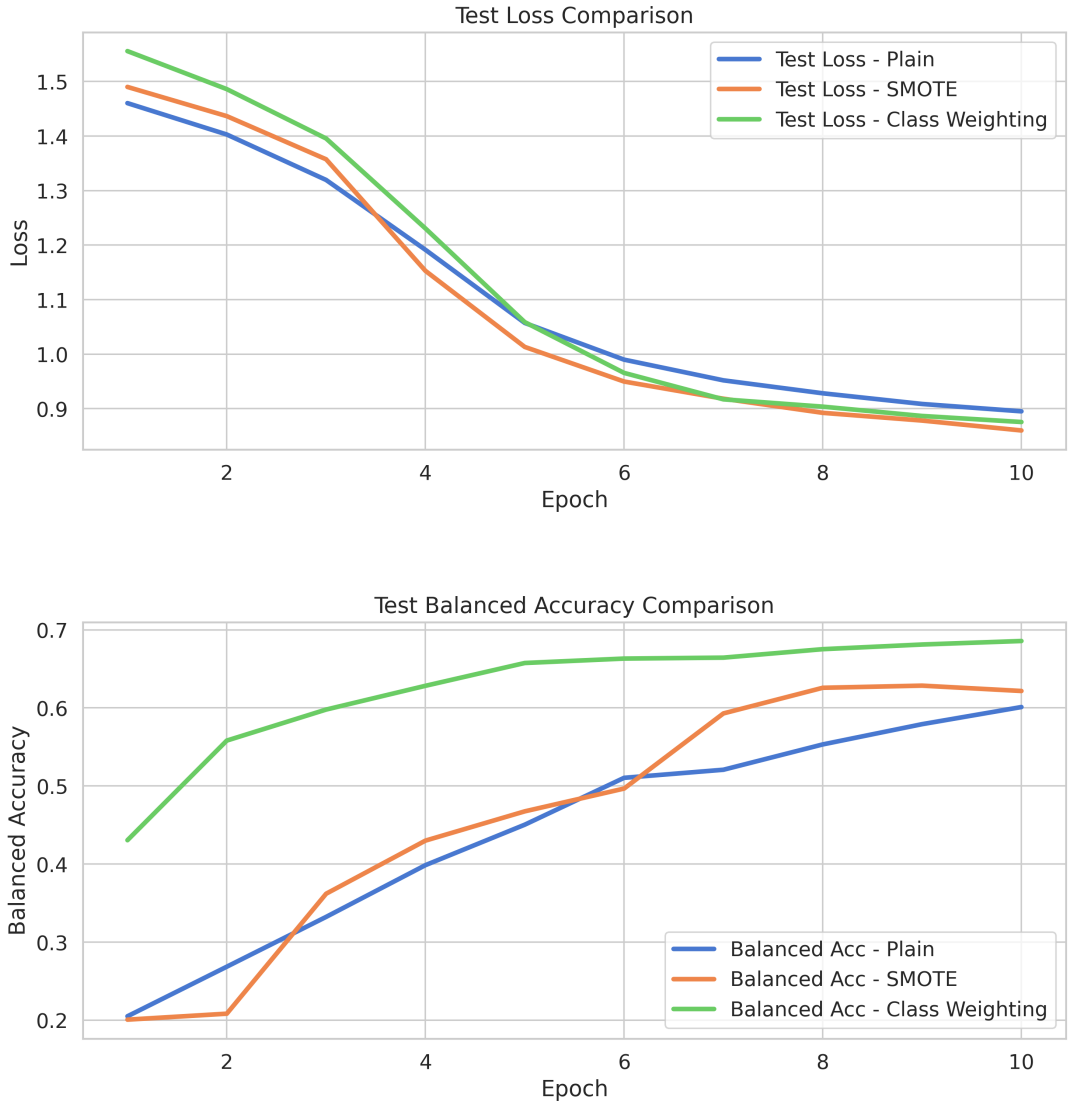


Figure 7.2: Bidirectional GRU with BioMedNLP embeddings: Test Loss and Balanced Accuracy over epochs for different strategies (no resampling, class weighting, and SMOTE). The experiments on the medical abstracts dataset were conducted with the following hyperparameters: batch size of 64, BERT embedding dimension of 768, hidden dimension of 256, dropout rate of 0.7, and a learning rate of 5×10^{-5}. The models were optimized using the Adam optimizer.

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural machine translation by jointly learning to align and translate*. arXiv preprint arXiv:1409.0473, 2014.

[2] J. Lee, W. Yoon, S. Kim, D. Kim, and C. H. So, *BioBERT: A pre-trained biomedical language representation model for biomedical text mining*, Bioinformatics, vol. 36, no. 4, pp. 1234–1240, 2020. https://doi.org/10.1093/bioinformatics/btz682

[3] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. *SMOTE: Synthetic Minority Over-sampling Technique*. Journal of Artificial Intelligence Research, 16:321–357, 2002.

[4] Simon Tong and Daphne Koller. *Support Vector Machine Active Learning with Applications to Text Classification*. Journal of Machine Learning Research, 2:45–66, 2001.

[5] Iker Huerga and Wendy Kan. *Personalized Medicine: Redefining Cancer Treatment*. Kaggle, 2017. https://www.kaggle.com/competitions/msk-redefining-cancer-treatment

[6] Tim Schopf, Daniel Braun, Florian Matthes. *Evaluating Unsupervised Text Classification: Zero-Shot and Similarity-Based Approaches*. Association for Computing Machinery, New York, NY, USA, 2023.

[7] Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L. E., & Brown, D. E. (2019). *Text classification algorithms: A survey*. Information, 10(4), 150. https://doi.org/10.3390/info10040150

[8] Uysal, A. K., & Gunal, S. (2014). *The impact of preprocessing on text classification*. Information Processing & Management, 50(1), 104–112. https://doi.org/10.1016/j.ipm.2013.08.006

[9] J. Pennington, R. Socher, and C. D. Manning, *GloVe: Global Vectors for Word Representation*, In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543, 2014. https://aclanthology.org/D14-1162/

[10] J. Snell, K. Swersky, and R. Zemel, *Prototypical Networks for Few-shot Learning*, NeurIPS 2017, pp. 4077–4087, 2017. https://arxiv.org/abs/1703.05175

[11] Y. Zhang, Q. Chen, Z. Yang, H. Lin, and Z. Lu, *BioWordVec, improving biomedical word embeddings with subword information and MeSH*, Scientific Data, vol. 6, no. 52, 2020. https://doi.org/10.1038/s41597-019-0055-0

[12] P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu, *Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling*, Proceedings of the 26th International Conference on Computational Linguistics (COLING), 2016, pp. 3485–3495. https://aclanthology.org/C16-1329

[13] C. Zhou, C. Sun, Z. Liu, and F. Lau, *A C-LSTM Neural Network for Text Classification*, arXiv preprint, 2015. https://arxiv.org/abs/1511.08630

[14] Y. Gu, R. Tinn, H. Cheng, M. Lucas, N. Usuyama, X. Liu, T. Naumann, J. Gao, H. Poon, *Domain-specific language model pretraining for biomedical natural language processing*, ACM Transactions on Computing for Healthcare (HEALTH), vol. 3, no. 1, pp. 1–23, 2020. https://arxiv.org/abs/2007.15779

[15] L. Prechelt, *Early Stopping — But When?*, In Neural Networks: Tricks of the Trade, Springer, 1998, pp. 55–69. https://doi.org/10.1007/3-540-49430-8_3