VRIJE UNIVERSITEIT AMSTERDAM

sincerius

Master Internship Report

Comparative Analysis of Traditional Machine Learning Models and BERT for Short Financial Text Classification

Author: Emiel Goldman (2560103)

First Supervisor:	Prof. Dr. Ger Koole
Second Reader:	Dr. Eduard Belitser
Organizational Supervisor:	Dr. Jan van Angelen

July 2, 2024

”Artificial intelligence is the new electricity.”
- Andrew Ng

Preface

This thesis was submitted for the Master's in Business Analytics program at Vrije Universiteit Amsterdam. It was written during my internship at SINCERIUS's Business Intelligence department, where I contributed to automating the organization's Financial Due Diligence sector.

SINCERIUS is faced with the task of manually labeling general ledger accounts, which can be time-consuming. My research aimed to automate this process by building machine learning models that automatically classify the financial texts within the general ledger accounts.

I would like to thank my supervisor, Professor Ger Koole from the Vrije Universiteit Amsterdam, for his responsiveness and for providing excellent guidance. I also thank my organizational supervisor, Dr. Jan van Angelen from SINCERIUS, for his assistance with planning and insights into the problem's financial aspects. I deeply appreciated the open and pleasant communication with both supervisors. Additionally, I would like to thank all the consultants at SINCERIUS for providing their expert opinions, which were invaluable in thoroughly understanding the problem. I also want to thank Dr. Eduard Belitser for serving as my second reader. Finally, I want to thank my parents, whose support made all of this possible.

Abstract

This thesis, conducted in collaboration with SINCERIUS, a financial due diligence company, aims to develop a model for automatically classifying ledger accounts, thereby reducing the many hours of manual work required for financial reporting each month. To achieve this, we evaluate the performance of traditional machine learning models—Logistic Regression, Random Forest, and Support Vector Machine—against the state-of-the-art pre-trained embedding models, BERT, and its Dutch variant BERTje. The task involves multi-class classification of short financial texts. We assessed the models across four different datasets using evaluation metrics such as accuracy, precision, recall, and F1-score and employed 5-fold cross-validation to ensure the robustness of our results. Our findings showed that BERT outperformed the other models on the two larger datasets, while Random Forest and BERTje have the best performances on the smaller datasets, highlighting BERT's sensitivity to dataset size. Additionally, we optimized the hyperparameters for all models across all datasets using the Optuna framework. Following this optimization, BERT produced competitive results on one dataset and demonstrated superior performance on the other three datasets, underscoring the importance of proper hyperparameter tuning for achieving optimal results with BERT. The final accuracies ranged from approximately 75% to 89%, significantly aiding consultants in classifying ledger accounts. We recommend exploring hybrid techniques that combine BERT with neural networks in possible future research. Another approach is to consider BERT ensemble models, which aggregate predictions from multiple variants of BERT to make final predictions.

Contents

1 Introduction **1**
- 1.1 Problem statement . . . 2
- 1.2 Contributions . . . 3
- 1.3 Thesis outline . . . 4

2 Literature Review **5**
- 2.1 Text classification . . . 5
- 2.2 Traditional machine learning models . . . 5
- 2.3 Pretrained Large-Scale Embedding Models . . . 6
- 2.4 Neural networks . . . 8
- 2.5 Feature engineering . . . 9

3 Data preprocessing **10**
- 3.1 Data overview . . . 10
- 3.2 Text preprocessing operations . . . 11
- 3.3 Feature extraction . . . 12
- 3.4 Feature engineering . . . 14
- 3.5 Class imbalance . . . 17

4 Methodology **18**
- 4.1 Traditional machine learning models . . . 18
  - 4.1.1 Logistic regression . . . 18
  - 4.1.2 Random Forest . . . 23
  - 4.1.3 Support Vector Machine . . . 25
- 4.2 Pre-trained Large-Scale Embedding Models . . . 27
  - 4.2.1 Word embeddings . . . 28
  - 4.2.2 Pre-trained models . . . 29
  - 4.2.3 BERT . . . 29

5 Experimental setup **35**
- 5.1 Performance evaluation . . . 35
- 5.2 Cross-validation . . . 36
- 5.3 Certainty scores and secondary predictions . . . 37
- 5.4 Hyperparameter tuning . . . 38

5.5 Optuna optimization 43

6 Results **45**
6.1 Profit & Loss statement 45
6.2 Balance sheet 48
6.3 Personnel expenses 50
6.4 Operational expenses 53
6.5 Optimal hyperparameters 55
6.6 Secondary prediction 57

7 Conclusions **59**
7.1 Main results 59
7.2 Hyperparameter tuning 60
7.3 Further research 60

References **62**

A Ledger account classes **69**

B Hyperparameter tuning results **71**
B.1 Logistic Regression 71
B.2 Random Forest 75
B.3 Support Vector Machine 79
B.4 BERT 84
B.5 BERTje 90

Chapter 1

Introduction

Artificial Intelligence (AI) is transforming industries, automating tasks, and improving efficiency across various sectors. In finance, managing vast amounts of data presents opportunities for AI. Most large financial and business services organizations have announced plans to utilize AI for virtually all statutory audits. This includes, for example, annual reports of companies that need to be approved by an accountant. This announcement highlights the rapid pace of technological integration within the industry. Despite the reduced workload facilitated by computer systems, repetitive tasks remain present. AI can be utilized to replace these tasks with automated systems.

Nowadays, most companies keep track of all their transactions, recording everything for a complete overview of their financial transactions. Moreover, all EU-based limited liability companies must prepare financial statements to monitor their business's health and provide a true and fair view of their financial performance and position [23]. This obligation is part of broader EU regulations that have been introduced to promote the convergence of accounting standards at the global level, ensuring consistent and comparable financial reporting across the EU.

All transactions within these financial statements are labeled to support financial analysis. This labeling process is facilitated by general ledger accounts, a core component of a company's accounting system. Each account within the general ledger represents a specific aspect of the company's financial activities. These accounts typically incorporate categories such as assets, liabilities, equity, revenue, and expenses. Organizing all transactions into these categories allows a clear view of the company's financial status and performance to be visualized, facilitating informed decision-making and strategic planning.

1.1 Problem statement

This study is conducted in collaboration with SINCERIUS, a financial due diligence company. SINCERIUS performs financial due diligence for several companies monthly, requiring consultants to classify the ledger accounts used in financial reporting. Accurate classification is essential for reliable financial analysis. Since a company typically has hundreds of ledger accounts, this classification process is time-consuming, often taking several hours of manual work for each new project. This study aims to develop a classification model that can automatically classify ledger accounts, potentially replacing several hours of manual labor for each new project.

The process of classifying ledger accounts relies on analyzing textual descriptions corresponding to the nature of the transaction. While text is a rich source of information, extracting meaningful insights from it is challenging due to its unstructured format. The descriptions of the ledger accounts are short, varying from one to ten words, which complicates the task of understanding the context of the piece of text. Moreover, general ledger accounts can be classified into various classes, making it a multi-class classification problem of short financial texts.

A general ledger account belongs to either of the two fundamental financial statements: the balance sheet or the Profit and Loss statement (P&L). The balance sheet is a snapshot of a company's financial position at a specific time, typically at the end of an accounting period, such as a quarter or a fiscal year. The P&L provides insights into a company's ability to generate profit and its operational efficiency and presents the financial performance of a company over a specific period. It summarizes revenues, costs, and expenses to determine the company's net income or net loss. Furthermore, the P&L statement has two ledger accounts which can be further divided into subclasses: 'Operational expenses' and 'Personnel expenses'. Classifying these subclasses are classification problems themselves, resulting in a total of four distinct datasets and classification tasks.

Automatic text classification can be achieved through rule-based methods or data-driven methods like machine learning. Rule-based methods categorize text into classes using predefined rules, demanding substantial domain knowledge. In contrast, machine learning–based approaches learn to classify text by leveraging observations from labeled data. Using pre-labeled examples as training data, a machine learning algorithm identifies inherent associations between texts and their labels.

Due to the complexity of classifying short texts, traditional programming methods do not provide a possible solution. Machine learning methods, however, hold promise in addressing this challenge, being able to learn context and adapt to the dynamic nature of text [7]. This research aims to develop such a machine learning model to classify general ledger accounts accurately. Therefore, various machine learning models with

distinct techniques were constructed and evaluated using several performance metrics.

Since the main goal is to find the best-performing model for classifying the ledger accounts, we will consider the current state-of-the-art model for text classification problems, BERT (Bidirectional Encoder Representations from Transformers), a pre-trained embedding model. To determine whether BERT does indeed show superior performance, we evaluate BERT along with three traditional machine learning models. Therefore, the central question addressed by this research is:

"What is the comparative performance of traditional machine learning models versus the state-of-the-art BERT model in classifying short financial texts?"

By comparing the performance of these models, we aim to identify which approach offers better accuracy and generalization to unseen data. This research provides useful insights into Natural Language Processing (NLP) and financial analysis, potentially guiding the use of automatic text classification in practical financial applications.

1.2 Contributions

This research makes several contributions to the field of automated general ledger account classification. To the best of our knowledge, this study pioneers the implementation of BERT, specifically for classifying general ledger accounts. BERT is known for its contextualized word embeddings and bidirectional processing and has proven to be exceptionally effective in capturing complex word relations within short textual descriptions [89]. This utilization of pre-trained embeddings enhances the model's ability to discern subtle nuances and context-dependent meanings in financial texts, making it particularly well-suited for the challenges posed by short-text classification in the financial domain. Therefore, we believe that BERT will outperform the traditional machine learning models.

Our study incorporates domain-specific knowledge into the models to enhance the accuracy of the classification process. Specifically, we integrate the account numbers corresponding with the ledger accounts as a domain-specific feature. These account numbers serve as indicators, providing contextual insights into the financial domain to which each ledger account belongs. By augmenting the textual descriptions with such domain-specific information, we aim to boost the classification accuracy of the models.

To ensure a fair comparison between traditional machine learning models and the pre-trained BERT model, it's crucial to optimize the models, giving each model an equal chance to perform at its best. Hyperparameter tuning ensures that all models are optimized to their fullest potential, reducing the risk of poor performances due to arbitrary default settings. Hence, we use Optuna, an optimization framework for fine-tuning ma-

chine learning hyperparameters. Optuna utilized more sophisticated optimization techniques than common hyperparameter tuning methods, such as random or grid searches. To our knowledge, Optuna has not previously been used to optimize the parameters of BERT.

Finally, we introduced a feature showing the certainty scores of the model's class predictions, indicating the likeliness of a prediction being correct. This helps the consultants at SINCERIUS filter out any wrongly predicted classes. Additionally, we added secondary predictions, which represent the second most likely classes according to the models. This approach offers a second suggestion, allowing the consultants to determine which is the correct class in cases where predictions are uncertain.

1.3 Thesis outline

The thesis is structured as follows. The introduction has set the stage by providing background information and introducing the problem and research question. Chapter 2 discusses the existing literature on automatic text classification. Following this, the datasets, data preprocessing steps, and feature engineering are presented in Chapter 3. We explain the workings of all our models in the 'Methodology' chapter. In Chapter 5, the experimental setup and the hyperparameter tuning methods are provided. The final results, together with the optimal hyperparameters, can be found in Chapter 6. Lastly, we summarize our findings and recommend further research in Chapter 7.

Chapter 2

Literature Review

The task of automatically classifying general ledger accounts remains relatively unexplored in the literature. Bardelli et al. [7] conducted experiments involving various algorithms and pre-processing techniques for automatic invoice classification. While recognizing the distinct challenges of invoice classification compared to ledger account classification, some of their methods proved applicable. The practice of selecting diverse pre-processing methods, a common approach in text classification [71], and fine-tuning methods have proven beneficial in this research.

2.1 Text classification

The primary task of labeling general ledger accounts involves categorizing the textual descriptions found on balance sheets and income statements. This task can be described as financial text classification, an area of significant application and research focus [29, 40]. Many researchers are interested in developing text classification applications, especially with recent developments in NLP and text mining [40].

With the increasing volume of text documents generated daily, the need for automated text classification has grown substantially. Beyond financial contexts, text classification finds utility in various applications such as web content management [35], search engines [67], email filtering [54], and biomedical texts [27].

2.2 Traditional machine learning models

Tong et al. [84] employed a Support Vector Machine (SVM) model for text classification, discovering relevant features through the SVM model. Moreover, they explored text classification by labeling words instead of entire documents, which requires document enrichment with numerous relevant words. However, a robust understanding of NLP and the utilization of high-dimensional representations is necessary for this, which

can be resource-intensive regarding memory and computing time. Consequently, this approach may prove inefficient for the limited time at hand and tasks like short text classification [28].

Ojala et al. [51] utilized different input columns from bank transactions for their financial statement classification problem. Their study compared accuracy across various companies on a trained neural network model, suggesting the viability of a generalized model for different companies. They deliberately excluded the description column containing free text or unstructured data. Nevertheless, they proposed that a machine learning model could still derive insights from the text by treating words as features within the input data.

Vink et al. [80] used several traditional machine learning models to classify ledger accounts, such as Naive Bayes (NB), SVM, Random Forest (RF), and xgboost. The RF model scored the highest accuracy in their first experimental run using a single train and test set. When using 5-fold cross-validation, however, the SVM model slightly outperformed the RF model. This finding aligns with the outcomes reported by Raicu et al. [59], who implemented an NB, SVM, RF, and a Logistic Regression (LR) model to classify Romanian ledger accounts. The SVM demonstrated superior performance compared to the other models, outperforming them by more than 2% in terms of accuracy. Bengtsson et al. [10] explored the performance of an SVM model and a Feed Forward Neural Network (FFNN) in classifying financial transactions. Interestingly, they trained the FFNN using Particle Swarm Optimization (PSO), a method demonstrated to train the model faster than the conventional backpropagation approach [31]. Their findings also revealed that the SVM model achieved the highest accuracy among the models evaluated.

2.3 Pretrained Large-Scale Embedding Models

Traditional machine learning models, such as LR, RF, and SVM, often perform less than embedding models for NLP tasks [48]. This is primarily because traditional models rely on manual feature extraction and cannot inherently capture the contextual meaning of words. They treat words as isolated tokens, ignoring the nuances and relationships between words in a sentence. This lack of understanding of context limits their ability to process and interpret the subtleties of language effectively [48].

In contrast, embedding models excel in NLP tasks because they capture semantic relationships between words by representing words as continuous vectors in a multi-dimensional space. These vector representations, known as embeddings, allow machines to understand better the contextual meaning of words based on their surrounding context in a given dataset. Embeddings are typically learned from large amounts of text data using unsupervised learning methods, enabling the model to understand language

patterns and relationships, which traditional models cannot achieve [32].

Latent Semantic Analysis (LSA), introduced by Deerwester et al. in 1989 [20], represents one of the earliest embedding models, employing a linear approach with fewer than 1 million parameters trained on 200,000 words. However, early neural language models initially underperformed compared to traditional machine learning models and faced limited adaptation in the field of text classification [48]. An example of such a neural language model is the feed-forward neural network based on 14 million words proposed by Bengio et al. in 2001 [8].

Wang and Deng [87] highlighted the challenge of obtaining descriptive features for short texts, impeding the classification process. To address this issue, one proposed solution is to augment information, as suggested by Yin and Shen in 2020 [91]. In their work, Yin and Shen introduced an innovative model incorporating information extracted from the text and data retrieved from an external knowledge base. They employed a Convolutional Neural Network (CNN) based text classification method, known as TextCNN, drawing inspiration from Kim's model from 2014 [37].

A major shift occurred with the development of significantly larger embedding models trained on extensive datasets. In 2013, Google introduced word2vec models, trained on 6 billion words, gaining widespread popularity for various NLP tasks [47]. Further development took place in 2017 when AI2 and the University of Washington created Embeddings from Language Models (ELMo), a contextual embedding model relying on a 3-layer bidirectional Long Short-Term Memory (LSTM) with 93 million parameters, outperforming word2vec by capturing contextual information [65]. In 2018, OpenAI constructed embedding models using the Transformer architecture, based on attention mechanisms [85].

In the same year, Google introduced BERT (Bidirectional Encoder Representations from Transformers), a pre-trained model consisting of 340 million parameters and trained on 3.3 billion words [22], also utilizing the transformer architecture. BERT represents the current state-of-the-art embedding model, demonstrating superior performance due to its ability to capture bidirectional contextual information.

González-Carvajal and Garrido-Merchan [30] employed BERT as a deep-learning model for supervised text classification and other NLP tasks, showing promising results. BERT utilizes transfer learning, where the model is pre-trained on a vast unlabeled corpus and then fine-tuned on labeled data for specific tasks. Unlike other transfer models, BERT can extract information bidirectionally from sentences, enhancing its understanding of context [22]. Empirical tests conducted by González-Carvajal and Garrido-Merchan concluded that BERT outperforms traditional machine learning algorithms on average NLP tasks and recommended its adoption as a default technique.

Concerns regarding the computational demands of pre-trained models like BERT led Sanh et al. to propose DistilBERT, a smaller and faster alternative that retains most of BERT's language-understanding capabilities through knowledge distillation [64]. Arslan et al. compared pre-trained models in financial multi-class text classification and found that DistilBERT outperformed BERT for certain datasets [6]. However, less optimistic findings have been reported in other studies, indicating that DistilBERT retains approximately 96% of language comprehension skills [75, 92].

Additionally, de Vries et al. [19] developed BERTje in 2019, a monolingual Dutch BERT model based on the same architecture and parameters as BERT but trained on a diverse dataset of 2.4 billion tokens. BERTje demonstrated superior performance to multilingual BERT on various NLP tasks, including part-of-speech tagging, named-entity recognition, semantic role labeling, and sentiment analysis. Considering the Dutch context of their dataset, it can be inferred that BERTje also outperforms BERT in this research, thereby warranting its inclusion alongside other mentioned models.

The trend of utilizing larger models and more extensive training data has persisted, with OpenAI's GPT-3 model containing 175 billion parameters [15], and the more recent GPT-4 model containing a massive 170 trillion parameters [39].

2.4 Neural networks

In addition to traditional machine learning methods and embedding models, text classification can also be addressed using deep learning techniques. According to Ali et al. [5], deep learning algorithms are often preferred over traditional ones. Deep learning has recently experienced significant advancements and successes, particularly in NLP [26].

Various types of neural network models exist, with three common examples being Feedforward Neural Networks (FFNN), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). FFNNs are multilayer perceptions and are among the most prevalent types of artificial neural networks [25]. In this architecture, neurons are organized in layers and fully interconnected, with unidirectional connections and devoid of cycles. As stated by Kowsari et al. [40], RNNs allow connections to form cycles, enabling the output from a layer of nodes to be reintroduced as input to that same layer. This cyclic structure allows information to be propagated through the network over time, offering advantages for text processing compared to the Feedforward approach by enabling the network to learn sequential patterns in the data [5].

CNNs perform two primary operations: convolution and pooling [26]. Convolution generates a feature map by element-wise multiplication of a matrix or filter, extracting features from the input. Pooling downsamples the feature map, reducing dimensional-

ity to create smaller, more manageable representations [5].

More recently, LSTM models have gotten more attention in the field of short-text classification. LSTM is a type of RNN architecture specifically designed to address the vanishing gradient problem, a common issue in traditional RNNs. LSTMs are particularly effective for tasks involving sequential data, such as NLP [9]. According to Zhao et al. [93], LSTM networks are effective in processing sequences because they consider the full context of text descriptions, making them superior to general RNNs in text classification. Their study demonstrated that the LSTM model yielded impressive results when classifying short financial descriptions across various languages.

2.5 Feature engineering

A range of complex features has been incorporated to enhance the text classification models. Examples include the integration of parts-of-speech and phrase information [41], the incorporation of syntax through explicit features and implicit kernels [56], and, for sentiment analysis, the inclusion of dependency tree features [49]. However, in a study by Wang & Manning [88], it was demonstrated that Bag of Words (BoW) and bigram features can be more effective than more complex alternatives for feature engineering. Therefore, we opted to utilize BoW for our traditional machine learning models in our research.

Li et al. [42] and Sun et al. [78] have applied the BERT model to text classification by generating auxiliary sentences and integrating domain-specific features. Both studies demonstrate exceptional performance on multi-class classification datasets, achieving state-of-the-art results. Incorporating domain-specific identifiers, such as account numbers, into classification models has significantly enhanced performance. For instance, Ojala et al. [51] integrated account numbers as features into their neural network for ledger account classification, leveraging domain knowledge effectively to improve classification accuracy. Similarly, Fan et al. [24] observed substantial improvements in their financial document classification model by incorporating domain-specific information, showing the importance of integrating domain-specific features in enhancing model performance.

Chapter 3

Data preprocessing

This chapter starts with an overview of the dataset, detailing its size and the classes represented in both the balance sheet and the P&L statement. It then discusses several data preprocessing steps essential for the NLP task at hand. The chapter concludes by exploring the feature engineering process.

3.1 Data overview

SINCERIUS, a data-driven financial due diligence company based in Amsterdam, provided the data for this research. Financial due diligence involves thoroughly examining a company's financial records, statements, and practices to assess its financial health, performance, and risks. They possess extensive labeled datasets from companies they conducted due diligence for, comprising multiple ledger accounts. Based on the corresponding descriptions, the consultants labeled these datasets manually, thereby facilitating the application of supervised machine learning models.

We have included 215 datasets of different companies, resulting in a highly diverse total dataset comprising 113.798 ledger accounts. After removing missing values and incorrect data, we were left with 98.720 accounts. Each ledger account either belongs to the balance sheet or the P&L statement, containing 42.232 and 56.488 accounts, respectively.

During this study, we identified instances where ledger account descriptions were distributed across multiple classes, which is suboptimal for the model as it impedes proper learning of the correct classes. Consequently, we removed all ledger accounts associated with a minority class linked to a specific description. This decision ensured the model assigns ledger accounts to the class with the highest probability. Following this removal operation, 41,187 ledger accounts remained on the balance sheet, while 54,195 were retained on the P&L statement.

The classes of the balance sheet can be seen in Appendix A.1. There are 18 classes on the balance sheet, with the class 'Tax' being the most frequent. The P&L statement contains 11 different classes, as seen in Appendix A.2. As discussed, the P&L statement contains the classes 'Operational expenses' and 'Personnel expenses', which contain six and four subclasses, respectively. The corresponding datasets are shown in A.3 and A.4. It can be seen that there is a strong class imbalance across all four datasets. Section 3.5 discusses the solution to this problem.

3.2 Text preprocessing operations

In this research, several common text preprocessing techniques are utilized. These techniques help prepare the text data for further analysis or modeling by standardizing, cleaning, and enhancing its quality, ultimately leading to better performance and more accurate results in NLP tasks [29]. Al-Hawari and Barham [33] compared the prediction accuracy of classification models trained on unprocessed datasets versus those preprocessed and cleaned. The results revealed a significant improvement in accuracy across all four classification models when using preprocessed and cleaned data, with an increase of approximately 20% to 30% in each experiment.

To ensure consistency and reproducibility of our models, we created the following data preprocessing pipeline:

1. **Lowercasing**
2. **Word Stemming**
3. **Tokenization**
4. **Removing Punctuation and Non-word Text**
5. **Removing Stop Words**
6. **Rejoining Stem Words**

Lowercasing Converting all abbreviations and capital letters to lowercase may seem like a straightforward preprocessing step, yet its importance is often underestimated. As Tabassum et al. [81] highlighted, this technique is particularly effective in sparse datasets. Consistency is ensured throughout the text data by converting all text to lowercase. This consistency is crucial as it ensures that words like "Word" and "word" are treated as the same token. Such uniformity is important because the model might interpret similar words as separate entities without them, leading to inconsistent results.

Removing Punctuation and Non-word Text Symbols and special characters are generally irrelevant in classifying text. Removing these punctuation and non-word characters helps clean the text data and focus on meaningful words.

Removing Stop Words Words such as "de", "en", "als" and "is" ("the", "and", "if", and "is" in English, respectively) commonly referred to as stopwords, typically hold little significance in NLP tasks, except in specific contexts. In text classification tasks, these stopwords are usually disregarded, and only keywords relevant to the topics are extracted. Identifying and removing these stopwords cleans the text and generally improves classification model results. However, it's important to recognize that in certain scenarios, negation words like "nee" and "niet" ("no" and "not" in English) play a crucial role in determining the context and intent of the sentence.

Tokenization Tokenization involves breaking text into individual tokens, such as words, characters, or punctuation. This step is essential for further analysis as it provides the basic units of text for processing. For example:

The textual description: "NLP is revolutionizing the finance industry!"
will be tokenized as:
"NLP", "is", "revolutionizing", "the", "finance", "industry", "!"

Word Stemming Stemming involves reducing words to their root or base form by removing suffixes while retaining the semantic meaning across various word forms. This can help reduce the vocabulary size and capture the essence of related words. For example, the word "playing" will be changed to "play" where "ing" is removed while the meaning of the word remains the same.

However, stemming may not always produce valid words and can sometimes result in ambiguous or incorrect stems [60]. For example, the term "coding" will be stemmed to "cod" which does not accurately represent the intended meaning in this context. Nonetheless, we incorporated word stemming into our data preprocessing pipeline for financial texts due to the diverse range of terms associated with financial instruments, markets, and activities, which are mostly stemmed accurately. Additionally, Singh et al. [76] demonstrated in their study that stemming financial texts simplifies text data representation and enhances performance. They observed an improvement of up to 10.7% in the average precision of the model.

Rejoining Stem Words Once the preprocessing steps are completed, the stemmed words are reintegrated into sentences. This practice is essential as preserving context significantly influences the classification of ledger accounts.

3.3 Feature extraction

Machine learning algorithms require numerical inputs. Feature extraction converts textual data into numerical representations, enabling algorithms to process and analyze

the text effectively. Moreover, text data can be high-dimensional, especially when dealing with large vocabularies or word embeddings. Feature extraction techniques help reduce the dimensionality of the data, making it more manageable for machine learning algorithms. Feature extraction aims to capture the essential characteristics or patterns present in the data, enabling the machine learning model to make accurate predictions or classifications.

In the context of NLP, feature extraction involves converting textual data into numerical representations, such as word vectors or document embeddings. Techniques like BoW, TF-IDF, and word embeddings are used to extract features from text data. These techniques transform words or documents into high-dimensional vectors, where each dimension represents a unique feature or aspect of the text.

Bag of words BoW is a popular and simple representation technique used in NLP and text analysis. This model is also known as the vector space model. In the BoW model, a document (or a text description) is represented as an unordered set of words, disregarding grammar and word order but keeping track of word frequency. The name "Bag of Words" suggests that you can imagine all the words in a document being thrown into a bag, and their order is ignored.

The BoW approach is advantageous not only for its independence from linguistic expertise but also for its ease of interpretation. However, a more precise analysis is necessary in certain cases, considering linguistic factors such as syntactic structure and semantic content. For instance, while the BoW representation would assign the same vector to the terms in the sentences "The student received critical feedback from the professor" and "The professor received critical feedback from the student", it is clear that the different word orders convey distinct meanings. Nonetheless, given our focus on classifying short financial texts, this issue is less prominent, leading us to choose BoW as our preferred method.

Term Frequency-Inverse Document Frequency One issue with BoW is its bias towards frequently appearing words, potentially inflating their importance [74]. However, some words may have high occurrence rates without contributing much information for classification or clustering tasks. Additionally, longer documents tend to receive higher scores than shorter ones, leading to reduced accuracy for the BoW model. To address these drawbacks, we turn to TF-IDF (Term Frequency-Inverse Document Frequency), which is a statistical measure utilized to assess the relevance of a word within a document (ledger account) amidst a collection of documents. This evaluation involves multiplying two metrics: the frequency of a word's occurrence within a ledger account (TF) and the inverse document frequency of the word across the entire dataset (IDF).

This measure finds broad application, particularly in automated text analysis, and proves highly beneficial for word scoring in machine learning algorithms employed for

NLP tasks. TF-IDF adjusts its scoring based on how often a word appears in a document and how common it is across all documents. This means that words like "dit", "wat" and "als" ("this", "what" and "if" in English) rank lower because they are common across many documents and carry less importance in any single document, indirectly assigning weights to the features.

Ali et al. (2021) highlighted TF-IDF as the most commonly employed feature extraction method for textual data [5]. In their study, TF-IDF outperformed other feature extraction techniques. The TF-IDF score for term t in document d from the document set D is given in Equation (3.1). The score is calculated as the product of the term frequency (TF) and inverse document frequency (IDF) [61]:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \tag{3.1}$$

Where $TF(t, d)$ is the number of occurrences of the term t in a document d and $IDF(t, d)$ is the number of occurrences of t in the document set D. Equation (3.2) shows the $IDF(t, d)$ calculation.

$$\text{IDF}(t, d) = \log\left(\frac{N + 1}{\text{DF}(d, t) + 1}\right) + 1 \tag{3.2}$$

$DF(d, t)$ is the document frequency, which is the number of documents d in the document set D containing term t, and N is the total number of documents. The "+1" term is added to the IDF equation to prevent division by zero and mitigate potential issues with terms that have zero document frequency.

Given the wide range of TF-IDF scores, it's advisable to employ feature scaling techniques [61]. Normalization, a commonly used approach, is particularly effective in this regard, as it transforms all term values onto a standardized scale ranging from zero to one. In this study, the L2 norm, also known as the Euclidean norm, is applied for normalization. The normalization of the TF-IDF value of a feature x is given in Equation (3.3).

$$x_{\text{norm}} = \frac{x}{\sqrt{\sum_{k=1}^{n} |x_k|^2}} \tag{3.3}$$

Here, x represents the TF-IDF value of a feature, k denotes all features where x appears, and x_{norm} signifies the normalized value of x.

3.4 Feature engineering

Feature engineering plays a crucial role in the success of text classification models. In addition to standard text preprocessing techniques such as tokenization, stemming, and stopword removal, incorporating domain-specific knowledge can further enhance

the performance of the classifiers.

Domain-specific knowledge refers to expertise or information specific to the subject being analyzed. This knowledge can be leveraged in various ways to enrich the feature space and improve classification performance. One feature that can serve as domain-specific knowledge is the ledger account number. This number determines the order of appearance on the balance sheet and the P&L statement. The account numbers typically consist of a 4-digit sequence, ranging from 0000 to 9999. Accounts falling under the 4000 range are commonly featured on the balance sheet, while those above 4000 are typically found on the P&L statement. Furthermore, the structure of these account numbers follows a hierarchical pattern, with the initial digit denoting the primary category of the ledger account, while the subsequent digits offer further granularity regarding the type of account. For instance, an account number starting with "4" often signifies an expense item on the P&L, with the following digit indicating specific types of expense items such as personnel expenses (40..), social securities (41..), or housing expenses (42..).

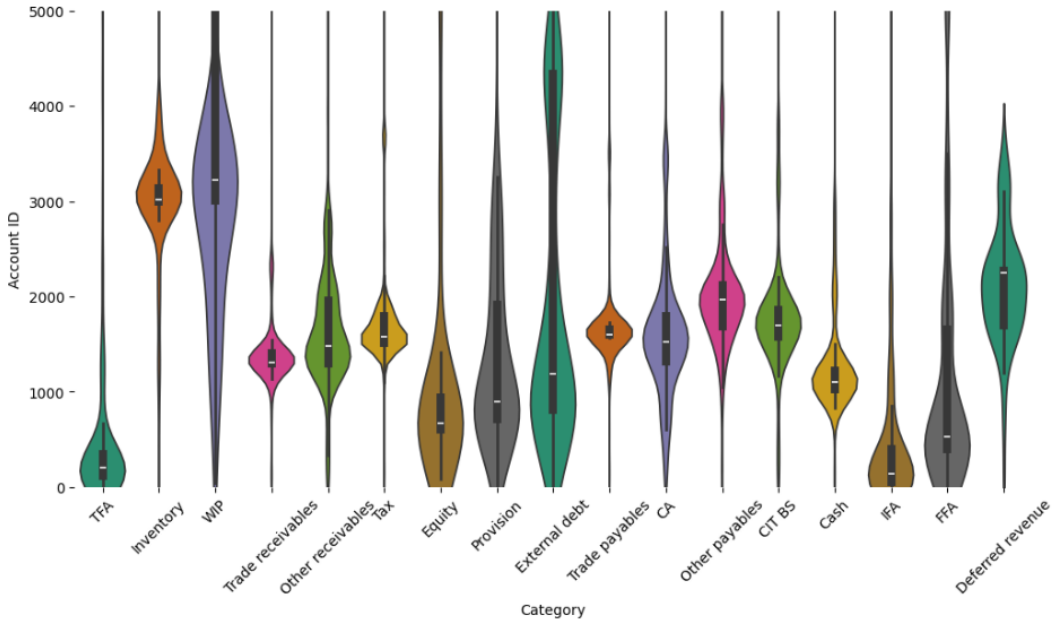

Figure 3.1: Account numbers on the Balance sheet.

Although this hierarchical system appears ideal for classifying ledger accounts, it's important to note that the companies assign these numbers manually, resulting in inconsistencies across the dataset due to human error, lack of standardized procedures, or variations in interpretation. Despite the lack of a one-to-one mapping, the account numbers still provide valuable insights for the classification task. The breakdown of the account numbers is shown per category in Figures 3.1 and 3.2. In Figure 3.1, TFA stands for Tangible Fixed Assets, WIP for Work In progress, CA for Current Account, CIT BS for Corporate Income Tax Balance Sheet, IFA for Intangible Fixed Assets and FFA for Financial Fixed Assets. In Figure 3.2, COS stands for Cost Of Sales, and CIT PL stands for Corporate Income Tax on the Profit & Loss statement.

The distribution of account numbers shows significant variations across all categories, with considerable overlap between them. However, many classes can be effectively excluded based solely on the account number. For instance, Figure 3.2 clearly shows that account numbers around 7000 or 8000 strongly indicate COS and Revenue, respectively. This suggests a potential model performance improvement by incorporating these numbers.

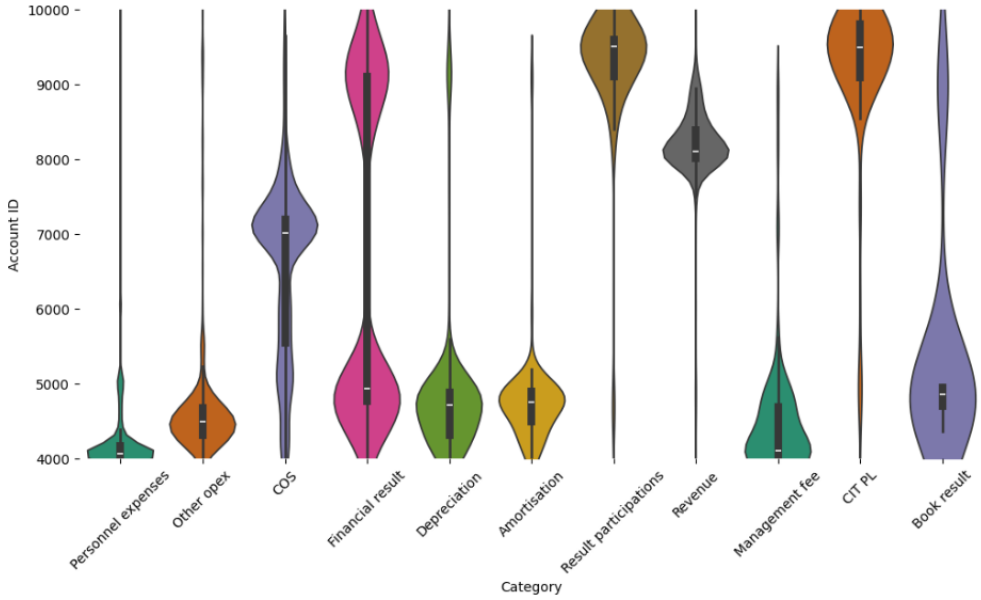

Figure 3.2: Account numbers on the Profit & Loss statement.

There are account numbers containing letters or more digits than four. These numbers are not considered since they are often based on a single company's policy regarding structuring the account numbers and, thus, do not contain any relevant information regarding the general classification of the ledger accounts. Therefore, only numbers within the range of 0000-9999 are concatenated to the descriptions.

The account numbers are incorporated into the model by concatenating the numbers to the text descriptions. An example of such concatenation is '4000 Brutoloon', which is Dutch for 'gross salary' and falls under the category of 'Personnel expenses'. The choice of text concatenation instead of adding a new feature to the model was based on the results found in the study of Pittara et al. [55], in which they found that the concatenation of semantic vectors offered the best performance boost.

3.5 Class imbalance

Handling data imbalance is crucial in multi-class text classification tasks to ensure that the model learns effectively from all classes [82]. We addressed data imbalance for our text classification problem by combining undersampling and SMOTE (Synthetic Minority Over-sampling Technique). Undersampling involves decreasing the number of samples from the majority class, a practice often avoided in research due to the loss of potentially valuable information from the majority class. On the other hand, oversampling can cause overfitting by excessively duplicating samples from the minority class, potentially causing the model to learn noise and specifics of the training data rather than true patterns.

To navigate these challenges, we investigated a combination of undersampling and oversampling techniques, a strategy applied in various studies [72, 77, 82]. This hybrid approach has demonstrated greater effectiveness than using oversampling or undersampling individually. By balancing the class distribution through this combination, we aimed to retain the essential information from all classes while mitigating the risks associated with traditional oversampling or undersampling methods. This approach uses the strengths of both techniques to achieve improved model performance and generalization capabilities, addressing class imbalance and producing a balanced dataset crucial for training classifiers.

SMOTE is frequently employed for oversampling due to its advanced capabilities in addressing class imbalance within machine learning. SMOTE addresses the imbalanced class distribution by generating synthetic samples for minority classes. Instead of directly replicating existing data points, SMOTE creates new synthetic examples that are strategically positioned within the feature space, close to existing minority class instances. By doing so, SMOTE aims to mitigate the risk of overfitting while effectively balancing class distribution, thus enhancing the model's ability to generalize to unseen data. This technique utilizes the inherent structure of the data to create meaningful synthetic samples, ultimately improving the robustness and performance of the classifier [52].

Chapter 4

Methodology

This chapter provides an overview of the models used in this research for classifying ledger accounts. We start by examining traditional machine learning models such as Logistic Regression (LR), Random Forest (RF), and Support Vector Machine (SVM). Subsequently, we discuss using pre-trained embedding models, specifically BERT and BERTje, which have shown remarkable capabilities in NLP tasks. Following exploring these models, we describe the hyperparameters corresponding to the models. Tuning these hyperparameters ensures the models are optimized for representative classification accuracy.

4.1 Traditional machine learning models

Despite the rise of pre-trained embedding models, traditional machine learning models remain relevant for several reasons. They often require fewer computational resources, making them more accessible when computational resources are limited. Additionally, these models tend to be more interpretable, offering insights into the decision-making process and facilitating the identification of influential features [3]. Models such as LR, RF, and SVM have shown their capability to identify patterns within textual data and accurately assign class labels to text documents [3, 68, 84].

4.1.1 Logistic regression

LR is a statistical method that can be used for binary classification problems and can be extended to handle multi-class classification. Despite its limitations, including the assumption of a linear relationship between input features and class labels, LR remains a common approach for text classification [70]. It offers simplicity in implementation and interpretation, making it practical for real-world applications. Moreover, LR can handle high-dimensional, sparse text data and can be regularized to prevent overfitting [70].

When applied to text classification, the goal is to predict the class of a given piece of text. First, one must represent the text as numerical features to utilize traditional machine learning models for text classification problems. Our approach uses the BoW representation, where each text is represented as a vector of word frequencies as discussed in Paragraph 3.3. The actual features used in our model are the weighted TF-IDF values, as discussed in Section 3.3.

Binary classification LR models the relationship between the features and the probability of the piece of text belonging to a particular class using the logistic sigmoid function. This is a mathematical function with a characteristic S-shaped curve given in Equation (4.1), where z is any real-valued number.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4.1}$$

The equation's inverse exponential term e^{-z} approaches 0 when z goes to infinity and increases without bound (approaches infinity) as z approaches minus infinity. As a result, the sigmoid function approaches 1 when z increases and 0 when z decreases, which can be seen in Figure 4.1. The output values between 0 and 1 make the sigmoid function ideal for modeling probabilities and binary classification tasks. The decision threshold of 0.5 serves as a dividing point for assigning text to specific classes in binary classification tasks. Moreover, it introduces non-linearity into the model, creating flexibility to model non-linear patterns in the data [70].

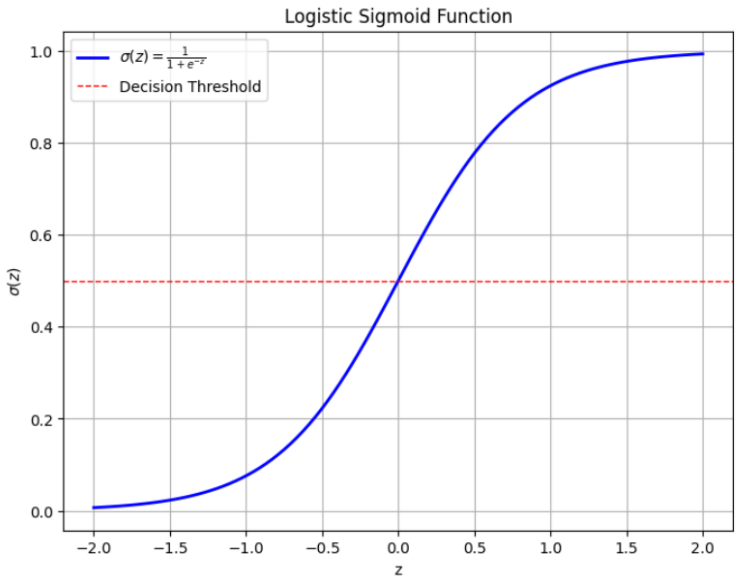

Figure 4.1: Logistic sigmoid function.

The LR model computes a weighted sum of TF-IDF values and applies the sigmoid

function as an activation function. This mapping transforms the weighted sum into a probability value between 0 and 1, representing the likelihood of belonging to a specific class. The final LR model equation is expressed as follows:

$$P(y = 1 \mid x; w) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}} \tag{4.2}$$

Where:

- $P(y = 1 \mid x; w)$ is the predicted probability of the class.
- w is the vector of weighted TF-IDF values.
- x is the binary input feature vector representing the presence of each word.
- $w^T x$ represents the dot product (weighted sum) of w and x.
- $\sigma(\cdot)$ is the logistic sigmoid function defined in Equation (4.1).

The model is represented in Figure 4.2. The input values $x_1, x_2, ..., x_n$ represent the presence of a word in the piece of text, where n represents the number of distinct words in the entire dataset. The weights $w_1, w_2, ..., w_n$ are parameters optimized by the model, representing the contribution of each word towards the classification task. These weights indicate the amount of information or relevance that each word adds to the classification process. The weights essentially determine how much each word influences the final classification decision. The summation process computes a single value by multiplying the sum of each input value by its corresponding weight.

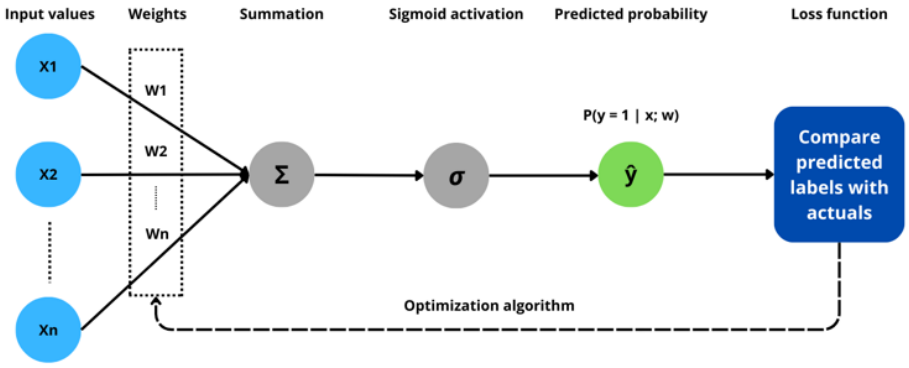

Figure 4.2: Logistic Regression model.

The calculated value is fed into the sigmoid function, which transforms it into a probability value between 0 and 1. Subsequently, the class prediction $\hat{y}$ is made based on this probability value, and all predictions are passed to the logistic loss function. The logistic loss function is a common choice for logistic regression because it is well-suited

to optimize the parameters of a model that predicts probabilities for binary outcomes [53]. The logistic loss function $\mathcal{L}(y, \hat{y})$ for the prediction of a single piece of text is defined as:

$$\mathcal{L}(y, \hat{y}) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right) \tag{4.3}$$

Where:

- y is the true class label (either 0 or 1) of the single piece of text.
- $\hat{y}$ is the predicted probability that the example belongs to class 1.

The overall logistic loss function $\mathcal{L}(w)$ is the average of individual loss terms of all predicted labels:

$$\mathcal{L}(w) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, \hat{y}_i) \tag{4.4}$$

Where:

- n is the number of texts in the train set.
- (x_i, y_i) represents the i-th training example with input features x_i and true label y_i.
- $\hat{y}_i = \sigma(w^T x_i)$ is the predicted probability for the i-th example using logistic regression.

Finally, the weights are updated in each iteration using an optimization algorithm (see Section 5.4).

Multi-label classification LR and SVM models are originally designed for binary classification tasks and do not inherently support multi-class classification. However, these binary classification algorithms can be adapted for multi-class problems by transforming the problem into a series of binary classification subproblems. Two common approaches for adapting LR and SVM for multi-class classification are the One-vs-Rest (OvR) and One-vs-One (OvO) strategies, described below.

One-vs-Rest:

- In OvR, we create K binary classifiers, each dedicated to distinguishing one class k from all other classes combined.
- During prediction, each binary classifier calculates a probability score for its corresponding class.
- The final prediction is made based on the binary classifier that assigns the highest probability score for its respective class.

One-vs-One:

- OvO decomposes the multi-class classification problem into $\frac{K(K-1)}{2}$ binary classification tasks, one for each pair of classes.
- Each binary classifier in OvO is trained to distinguish between a specific pair of classes.
- During prediction, each binary classifier votes for one of the two classes.
- The predicted class label for a new instance is determined based on the class that receives the most votes across all binary classifiers.

Regarding dataset characteristics, OvR effectively handles class imbalances because each binary classifier focuses on distinguishing one class from all others. This approach can be particularly advantageous when dealing with datasets where certain classes are more prevalent than others. Additionally, OvR scales well with many classes compared to OvO, as it requires fewer binary classifiers to be trained, making it more efficient and feasible for high-dimensional classification tasks. On the other hand, OvO is preferred when dealing with complex decision boundaries and intricate class relationships. By training classifiers for each pair of classes, OvO can better capture nuanced relationships between classes and handle scenarios where classes might exhibit overlapping characteristics or dependencies. Since it is not obvious which approach will perform better for our problem, both approaches are considered in the hyperparameter tuning step.

4.1.2 Random Forest

RF is a powerful ensemble learning method commonly used for classification and regression tasks. It belongs to the family of tree-based models and is known for its robustness, flexibility, and ability to handle high-dimensional datasets effectively [13].

decision tree A decision tree is a graphical representation of a decision-making process depicted in a tree-like structure. It consists of nodes that signify specific conditions, branches representing the presence of a word, and leaf nodes that denote the final classification. An example of a decision tree is visualized in Figure 4.3.

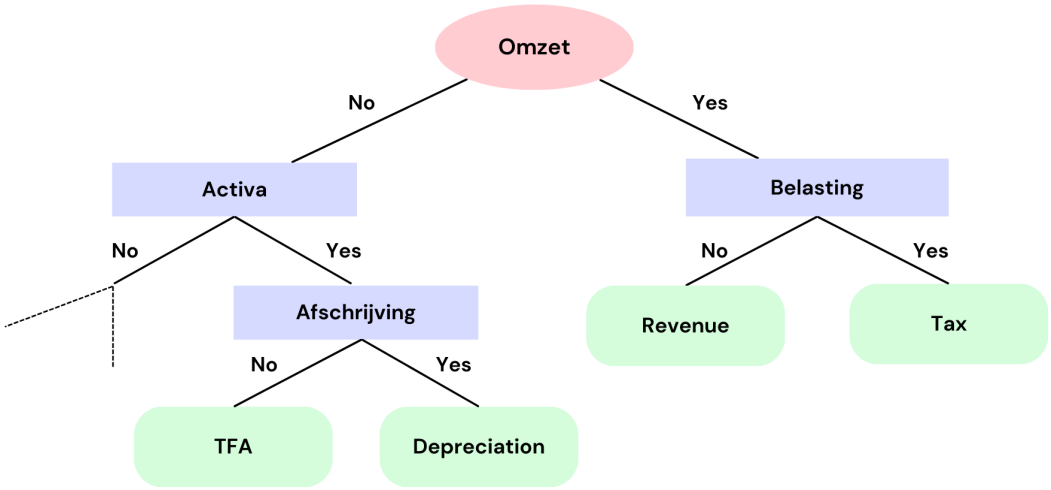

Figure 4.3: An example of a Decision Tree.

In this example, the decision tree begins with the root node labeled 'Omzet' (Dutch for Revenue), representing the best data-splitting feature. As we move down the tree, each internal node represents a test on a specific feature (e.g., the presence of a specific word in the piece of text), which divides the dataset into smaller subgroups based on the feature's value. Here, 'Activa', 'Belasting', and 'Afschrijving' are Dutch for assets, tax, and depreciation.

The process continues recursively, with each split maximizing the homogeneity of the target variable within the resulting subgroups. This means that after each split, the tree aims to make the resulting groups as pure as possible concerning the target class. Ultimately, as we traverse the tree from the root to the leaves (green blocks), we can classify a new instance based on its path through the tree. For instance, if a ledger account does not contain the words 'Omzet' and 'Afschrijving' but does contain 'Activa', the decision tree model classifies the instance as TFA (Tangible Fixed Assets).

The tree is built by selecting the best feature to split on at each node. The splitting

process continues recursively until the tree is fully grown or a stopping criterion is met. Possible stopping criteria are discussed in Paragraph 5.4.

Optimal feature selection The best feature for splitting the data is typically determined using a criterion called "Gini impurity" or "information gain." The Gini impurity is the default setting for our RF model. It measures how often a randomly chosen element from the dataset would be incorrectly classified if it were randomly labeled according to the class distribution in the subset.

Information gain measures the target variable's entropy (or uncertainty) reduction after splitting the dataset on a particular feature. Features that result in higher information gain are preferred because they lead to more significant reductions in the uncertainty of the target variable [45].

Random Forest ensemble RF expands the concept of decision trees by creating multiple trees during training and combining their predictions to enhance accuracy and generalizability [70]. In an RF model, each tree is trained using a subset of the training data and considers only a subset of the input features (chosen randomly), which leads to varied class predictions among the trees. This deliberate introduction of randomness helps in reducing overfitting and encourages diversity among the models in the ensemble [13]. When building the RF model, each tree is trained independently to predict the target class of a given input. The final predictions of the RF model are then made through majority voting or probability averaging.

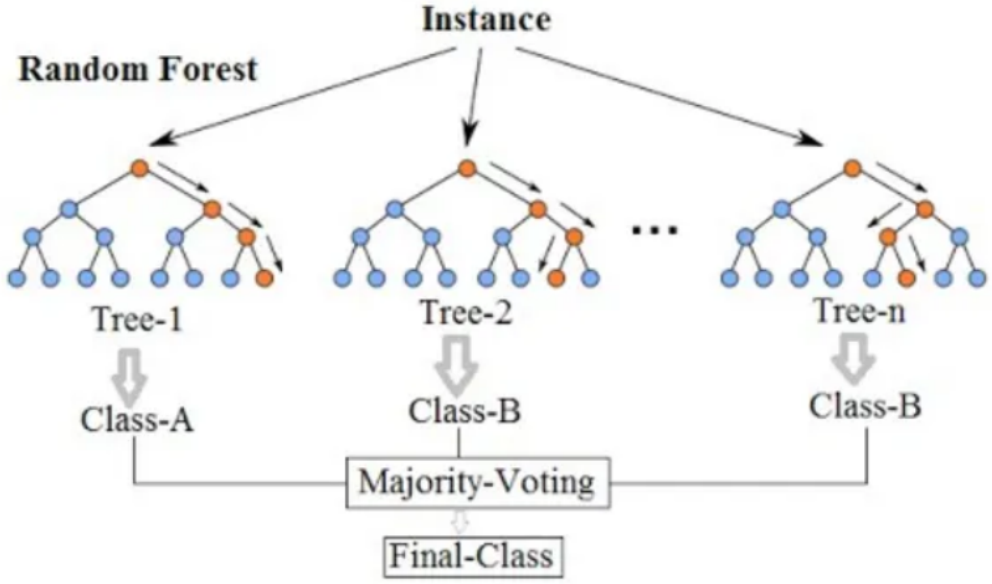

Figure 4.4: Majority voting in a Random Forest model [38].

When all decision trees are built and trained, each tree 'votes' for a class label. In majority voting, the class that receives the most votes is considered the final prediction of the RF. Averaging in RF models is typically used for regression tasks, where each tree predicts a value for the target variable. The predicted values from all the trees are then averaged to obtain the final prediction. While direct averaging of class labels is impossible, an alternative approach known as probability voting can be used in classification tasks. Probability voting involves averaging the predicted probabilities assigned to each class by the individual trees. This can provide a more nuanced and probabilistic approach to combining predictions, where the final class label is determined based on the highest average probability across all trees. Figure 4.4 shows a simplified version of the RF model. Here, n stands for the number of decision trees.

4.1.3 Support Vector Machine

SVM is another powerful supervised learning model used for classification and regression tasks. Like LR, SVMs are particularly effective for binary classification but can be extended to handle multi-class problems using OvR or OvO. The primary goal of SVMs is to determine an optimal hyperplane that effectively separates different classes of data points within a high-dimensional space. This involves identifying a hyperplane capable of distinctly segregating data points associated with distinct classes [79].

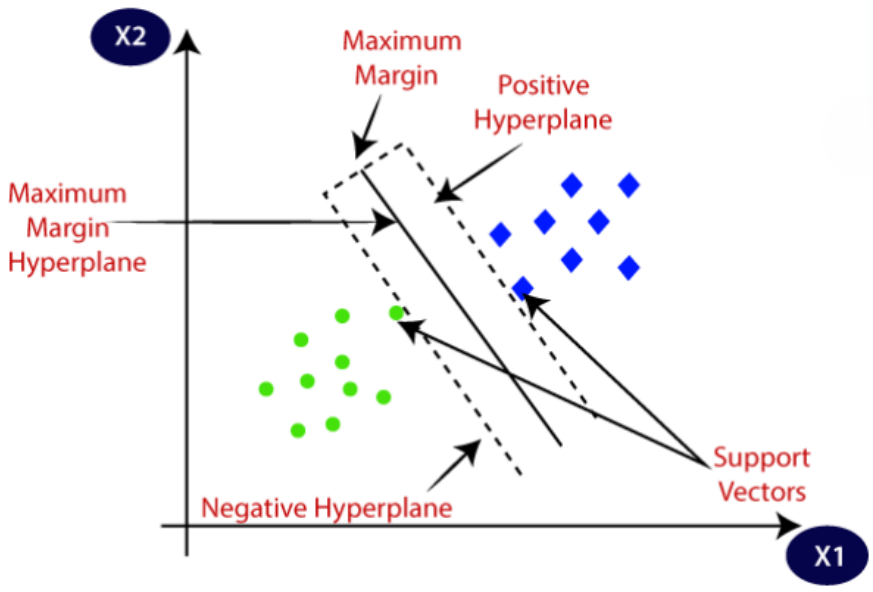

Figure 4.5: Simplified Support Vector Machine model [63].

In an n-dimensional space (where n represents the number of distinct words), a hyperplane is essentially a flat subspace of dimension $n-1$. For instance, within a 2D space (comprising two features), a hyperplane manifests as a straight line; within a 3D space

(with three features or distinct words), it assumes the form of a plane. To visualize this concept, consider Figure 4.5, showing a simplified example of a hyperplane within a 2D space, dividing two distinct classes.

The characteristic of SVM lies in its prioritization of maximizing the margin, which signifies the perpendicular distance between the hyperplane and the nearest data points, known as support vectors, for each class. The positioning of the hyperplane is directed to optimally separate the classes by achieving the widest margin feasible. SVM aims to find the hyperplane that maximizes this margin because a larger margin implies better generalization and robustness of the model against unseen data [36]. Since the objective is to maximize the margin between the support vectors, SVM mainly relies on the support vectors to define the classifier. This focused approach means that SVM does not concern itself with all observations but rather leverages the critical support vectors to establish the optimal hyperplane.

When dealing with perfectly linearly separable data, a hyperplane can effectively partition the data with a single line in a 2D space. However, real-world datasets, particularly those with numerous features, are often complex and nonlinear. This complexity poses challenges for linear separation, as seen in our text classification problem.

To address this challenge, we employ the "kernel trick", a technique that handles nonlinear classification problems. The kernel trick works by mapping data points into a higher-dimensional space where linear separation becomes possible. The kernels we consider for our SVM model are discussed below, along with their corresponding formula.

- **Linear Kernels**: Define the dot product between the input vectors in the original feature space. The linear kernel is given in Equation (4.5). Where K stands for the kernel, and x and y are the input feature vectors.

$$K(x, y) = x^T \cdot y \tag{4.5}$$

- **Polynomial Kernels**: Introduce nonlinearities using polynomial transformations, allowing for flexible decision boundaries. The polynomial kernel is defined in Equation (4.6), where c is a constant and d is the polynomial degree.

$$K(x, y) = (x^T \cdot y + c)^d \tag{4.6}$$

- **Radial Basis Function (RBF) Kernels**: Utilize Gaussian functions to capture complex relationships and local similarities. The RBF kernel equation is shown below:

$$K(x, y) = e^{(-\gamma \|x-y\|^2)} \tag{4.7}$$

Here, γ is a parameter controlling the Gaussian function width, which determines the degree of nonlinearity in the decision boundary, and $\|x - y\|$ is the Euclidean distance between the two input vectors.

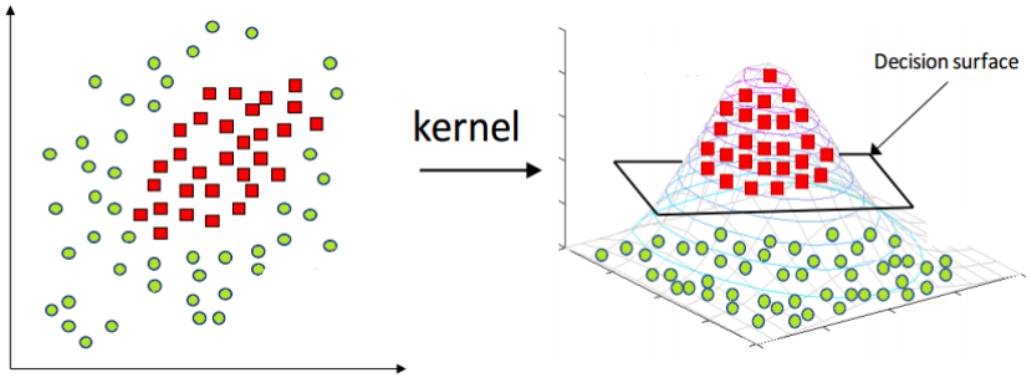

Figure 4.6: Kernel example [63]

To explain the functionality of a kernel, let's consider the scenario depicted in Figure 4.6. On the left, we see a non-linearly separable dataset existing within a 2D space. The challenge lies in finding a linear boundary (hyperplane) that separates the red squares from the green circles. To tackle this problem, a kernel transforms the original 2D dataset into a higher-dimensional 3D space. This transformation is executed by mapping each data point to a new coordinate system. During this mapping process: The kernel elevates the positions of the red squares along a newly created axis in 3D space. Conversely, the positions of the green circles are shifted downwards along the same axis. This transformation allows for linear separation within 3D space. Consequently, a hyperplane (a plane in 3D) can be introduced to effectively segregate all data points along distinct regions, thereby achieving the desired linear separability.

4.2 Pre-trained Large-Scale Embedding Models

Although Deep Learning models have achieved state-of-the-art performance in various NLP tasks, they require training from the ground up, demanding substantial datasets and days to converge [34]. In response to this challenge, recent years have witnessed a revolutionary shift in text classification and various NLP tasks, thanks to pre-trained large-scale embedding models. Examples like BERT, built upon transformer architectures, have been pre-trained on extensive textual datasets.

4.2.1 Word embeddings

Word embeddings play a crucial role in capturing the semantic meaning of words within the context of a sentence or a document. In word embeddings, the representation of each word is dependent not only on its own characteristics but also on the surrounding words in the sentence.

These representations are made by transforming words into sets of real numbers, forming vectors that position words in an n-dimensional space. This spatial arrangement in a high-dimensional vector space tries to encapsulate the meanings of words, allowing machines to interpret and process textual information more effectively [69].

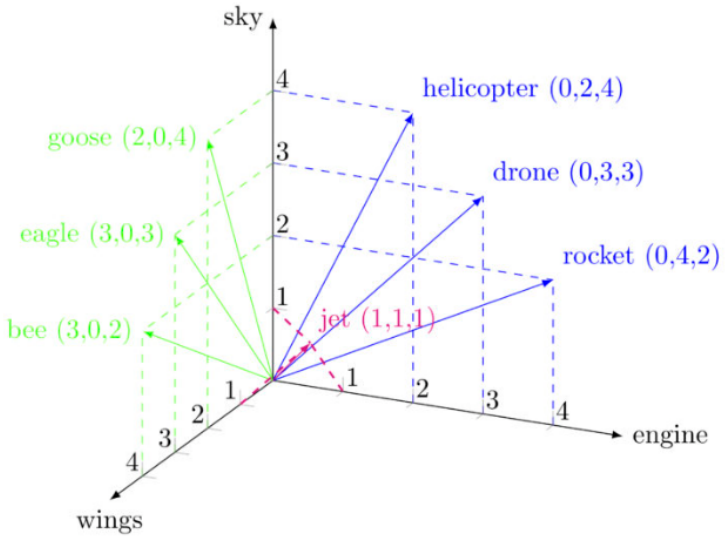

Figure 4.7: Word embedding example [21].

Figure 4.7 shows a simple representation of a word embedding using three features and a 3-dimensional vector space. The position of each word in this figure is determined by its vector representation in the embedding space. This way, words that are semantically similar or share common contexts will be closer to each other. The values in the figure (e.g., the numbers 0, 1, 2, 3, 4) represent the strength of association or similarity between each word and context. Higher values indicate stronger similarity, while lower values indicate weaker similarity. This way, we can observe semantic relationships between the words. For example, words like "bee" and "eagle" might be closer together due to their associations with "sky" and "wings," while "helicopter" and "jet" might be closer due to their association with "engine". Words with similar meanings are located closer together in this vector space. This approach enhances computational efficiency and enables machines to derive meaningful insights from textual data, transforming raw text into a structured format beneficial to machine learning algorithms [55].

4.2.2 Pre-trained models

State-of-the-art NLP models often use pre-trained word embeddings trained on large amounts of text data. These pre-trained word embeddings are a type of transfer learning, a machine learning technique where a model is trained on one task, and a dataset is reused for another related task. Instead of starting the learning process from scratch, transfer learning leverages knowledge gained from solving one problem and applies it to a different but related problem.

Pre-trained word embeddings represent a form of unsupervised learning where a model is trained on a large corpus of text to generate dense vector representations of words. These embeddings encode semantic information about words based on their contextual usage within the text. Using pre-trained word embeddings, the vast amount of existing texts can be utilized in tasks with small labeled datasets, achieving high accuracy with limited data [57]. Pre-trained embedding models have proven effective in enhancing various NLP tasks, outperforming traditional machine learning and deep learning techniques [17, 22, 34, 57].

4.2.3 BERT

BERT is a deep learning approach that utilizes transformers for pre-training in NLP. It was developed in 2018 by Jacob Devlin and other Google researchers [22]. Since its release, BERT has demonstrated state-of-the-art performance across various NLP tasks, getting much attention in the NLP community.

Transformer architecture The architecture of BERT's model is a multi-layer transformer inspired by the original design detailed by Vaswani et al., in 2017 [85]. The transformer uses the encoder-decoder structure, a fundamental architecture in neural networks, particularly for NLP tasks [16]. In this architecture, the encoder transforms an input sequence of words, in our case $(x_1, ..., x_n)$, into a sequence of continuous representations $(z_1, ..., z_n)$ using embeddings. The decoder then generates the output sequence of symbols $(y_1, ..., y_m)$ based on the encoded information from the encoder.

The transformer structure can be seen in Figure 4.8. Where the left and right sides represent the fully connected encoder and decoder, respectively. Where N is the number of identical layers, which is set to 6 for both the encoder and decoder. The encoder consists of two sub-layers, a multi-head attention layer, and a fully connected feed-forward network. The decoder has a third layer, performing masked multi-head attention as the first step. Each mechanism is followed by a simple normalization operator.

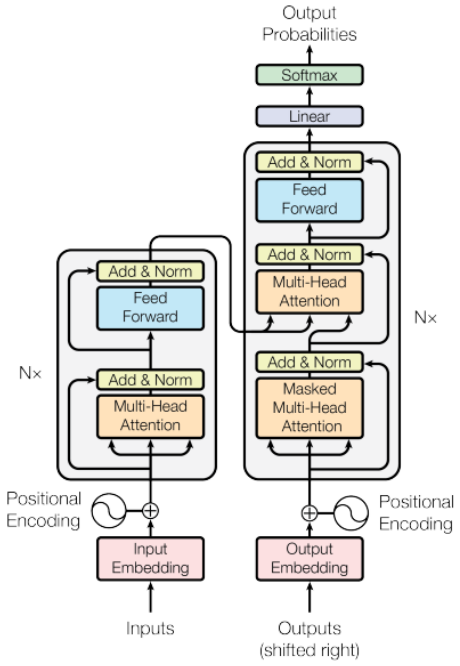

Figure 4.8: The transformer model architecture [85].

The attention mechanisms allow the model to focus on different parts of the input sequence in parallel. It maps a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is calculated as a weighted sum of the values, where each weight is determined by a compatibility function that measures the similarity between the query and the corresponding key [85].

A single-head attention function is computed in parallel, concatenating the vectors of the queries, keys, and values into matrices Q, K, and V, respectively. The matrix of outputs is computed as a scaled dot-product and is given in Equation (4.8).

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.8}$$

Where d_k is the dimension of the query and key vectors, and T indicates a transpose operator, which is necessary for matrix multiplication. Finally, multi-head attention is the concatenation of each single-head, resulting in the structure given in Figure 4.9, resulting in Equation (4.9). Where W^O is the learned linear projection matrix for the concatenated output, and h is the number of parallel attention heads in the concatenation, which is set to 8.

$$\text{Multi-Head}(Q, K, V) = \text{Concatenate}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{4.9}$$

In masked multi-head attention in the decoder part, future words are 'masked,' preventing the model from accessing these words, effectively creating a causal model. This

masking is crucial for preserving the temporal order and coherence of the output [58]. Additionally, it significantly accelerates training by capturing the relationship between each token and its immediate successor within a single data structure.

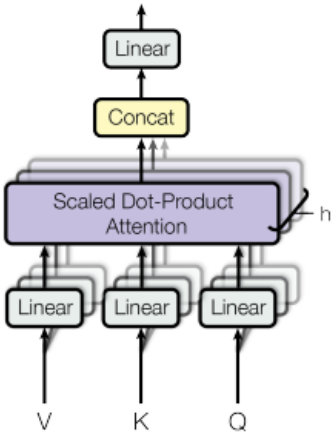

Figure 4.9: Multi-head attention mechanism [85].

BERT architecture BERT consists of 12 transformer blocks, a hidden layer of size $h = 768$, and 12 self-attention heads. Due to its input representation, which consists of a token sequence that can represent either a single sentence or multiple sentences combined, BERT can handle various downstream tasks. This token sequence is generated using the WordPiece embeddings, introduced by Wu et al. [90], with a vocabulary of 30,000 tokens. In BERT, each sequence starts with an initial classification token denoted by ([CLS]). Sequences containing multiple sentences are separated with a separation token ([SEP]). A general overview of the pre-training structure is shown in Figure 4.10. Here, the token C is the final hidden vector of the classification token, E denotes the input embeddings, and T_i is the final hidden token of the i^{th} input token. BERT has two main stages: pre-training and fine-tuning. The model trains on unlabeled data across various tasks in the pre-training phase. In the fine-tuning stage, BERT is initialized with the pre-trained parameters, and all parameters are tuned using our labeled dataset specific to the classification of the ledger accounts. An advantage of the BERT model is its pre-training architecture, which applies to various downstream tasks, resulting in similar structures across distinct tasks [22]. Therefore, there is no need to adjust the architecture to deal with our text classification task.

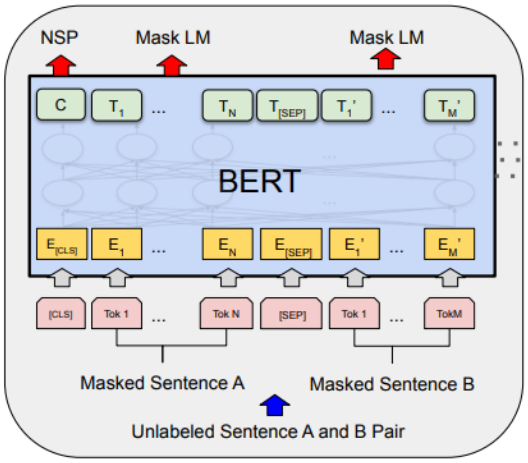

Figure 4.10: BERT pre-training architecture [22].

Pre-training The pre-training process utilizes the BooksCorpus and English Wikipedia, which consist of 800 million and 2500 million words, respectively, for a total of 3.3 billion words. The lists, tables, and headers are not included in the Wikipedia dataset, and only text passages are extracted. The method follows the procedure of existing literature on language model pre-training [22]. The pre-training comprises two unsupervised tasks: Masked Language Modelling (MLM) and Next Sentence Prediction (NSP). MLM, commonly known as the Cloze task [83], involves randomly masking some percentage of the tokens in the input text and training the model to predict these masked tokens based on the context provided by the surrounding tokens. Below is an example to illustrate this:

- **Original Sentence:** "The smart young student solves the complex problem."
- **Masked Sentence:** "The smart young [MASK] solves the complex problem."
- **Model Prediction:** The model analyzes the surrounding context ("The smart young" and "solves the complex problem.") and predicts that the masked word is "student."

During training, the model is provided with sentences containing masked words. The model learns to fill in these masked words by understanding the context of the words around them. This representation combines the left and right context, allowing for bidirectional learning and showing state-of-the-art results compared to traditional left-to-right language models [22].

NSP is a training task in which the model is provided with pairs of sentences. The goal is to predict whether the second sentence logically follows the first one. The C-token in Figure 4.10 is used for the NSP task. It is labeled as 'IsNext' 50% of the time, containing the actual sentence following the first one, and 'NotNext' the other

50% of the time, containing a random sentence from the training data. Below are two examples, one containing a sentence pair that logically follows each other and one that does not.

- **True example:**
  - **Sentence 1:** "I prepared well for my big presentation."
  - **Sentence 2:** "The presentation went smoothly without any problems."
  - **Label:** IsNext.
- **False example:**
  - **Sentence 1:** "I prepared well for my big presentation."
  - **Sentence 2:** "My bike broke down on the way to work."
  - **Label:** NotNext.

Fine-tuning Due to the self-attention mechanism within the Transformer architecture, several downstream tasks can be fine-tuned simply by adjusting the inputs and outputs accordingly [22]. For our text classification task, we integrate the preprocessed short financial texts (the ledger accounts) as inputs and the corresponding labels as outputs (as the [CLS]-token representation) into BERT. This allows us to fine-tune all parameters end-to-end. The fine-tuning process is given below:

- Initialize the model with pre-trained weights.
- Feed the tokenized input texts into the model.
- Pass the output representation of the [CLS]-token through the classification layer.
- Measure the difference between the predicted class probabilities and the actual class labels.
- Train the model by updating the weights, using backpropagation and an optimization algorithm (Adam, discussed in Section 5.4).
- Repeat this process for several epochs, adjusting the model parameters to minimize the loss of the training data.

BERT variations Since its release, researchers have been studying and refining BERT, leading to several model variations. We have experimented with DistilBERT (a distilled version of BERT [64]), RoBERTa (Robustly optimized BERT Pre-training approach [44]), and BERT Large. DistilBERT is a smaller and faster model trained by distilling BERT, RoBERTa adjusts key hyperparameters by eliminating the next-sentence pre-training objective and opting for significantly larger mini-batches and learning rates during training, and BERT Large is a larger version of BERT with more encoders, self-attention heads, and parameters.

The modified models highlighted in their respective studies showed promising results [22, 44, 64]. However, in our experiments, they consistently underperformed. Only BERT Large demonstrated comparable results in certain cases, but its extensive training time exceeded our available computing resources. These outcomes could be attributed to differences in the datasets they used, which mainly contain larger English texts, whereas our datasets mainly contain short Dutch texts. This suggests that the original BERT model provides the most reliable generalization compared to its alternatives. These findings align with several studies regarding the performance of BERT and its alternative models [6, 62, 86].

Nonetheless, we did include the monolingual Dutch variant BERTje [19]. This model was developed using the same architecture and parameters as BERT, except for some modifications in the pre-training procedure for the MLM and NSP tasks. BERTje is trained using a different dataset of several high-quality Dutch text collections. We chose this modified model because our datasets mainly consist of Dutch ledger accounts. Therefore, we anticipated that the BERTje model would outperform BERT.

Chapter 5

Experimental setup

This section discusses the experimental setup of this research. First, we describe our performance evaluation techniques to assess each model's performance in classifying the ledger accounts. Next, we discuss cross-validation, a technique that assesses the performance and generalizability of the models. Following this, we present two methods to aid consultants in their decision-making regarding the prediction of the ledger account class. The first method provides certainty scores associated with the predicted classes, and the second method incorporates a secondary prediction. This approach facilitates more informed decision-making, enabling consultants to make effective judgments based on the model's outputs. Finally, we explain the Optuna hyperparameter tuning framework.

5.1 Performance evaluation

The most common and comprehensive performance metric in machine learning is accuracy. However, accuracy can be misleading in datasets with imbalanced classes. For instance, in a dataset with 90% of instances belonging to one class, a model that always predicts the majority class would achieve 90% accuracy, despite having no predictive power for the minority class. Furthermore, Accuracy does not differentiate between types of errors. It treats false positives and false negatives equally. Finally, accuracy does not provide information on how well the model performs on individual classes, which is important in our case since we want to identify which classes the model tends to predict wrongly. Therefore, alongside the accuracy, we use the metrics precision, recall, and the F1-score; this way, we offer better insights into the generalizability of the models. Explanations of the four different evaluation metrics are presented below, alongside their corresponding equations.

Accuracy The first metric is accuracy, defined in Equation (5.1), where TP stands for True Positives, TN for True Negatives, FP for False Positives, and FN for False Negatives. The accuracy calculates the ratio of correctly predicted instances to the total

number of instances. It represents the probability that the model correctly predicts the suitable class.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Precision The precision score, described in Equation (5.2), evaluates the proportion of true positives among all positive predictions made by the model. This metric indicates the model's reliability when it predicts a particular class [75].

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.2}$$

Recall The recall score, defined in Equation (5.3), measures the model's ability to identify all relevant instances within a dataset.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.3}$$

F1-score The final metric is the F1-score, given in Equation (5.4), which combines precision (PS) and recall (RC) into a single metric by calculating their harmonic mean. This score balances precision and recall, offering a comprehensive measure of the model's performance [75].

$$\text{F1-score} = 2 \times \left(\frac{PS \times RC}{PS + RC} \right) \tag{5.4}$$

5.2 Cross-validation

To evaluate the models robustly, we utilized 5-fold cross-validation. This technique partitions the dataset into five subsets, ensuring each fold contains a balanced representation of all classes. Figure 5.1 visualizes the concept of cross-validation. It can be seen that the data is split into five folds. The model is trained on four folds and validated on the remaining one fold. This process is repeated five times, each time with a different fold used as the test set and the remaining folds as the training set. After the five iterations, the performance metrics (accuracy, precision, recall, F1-score) are averaged to produce a single estimate of the model performance. By evaluating the model on multiple subsets of the data, cross-validation provides a more reliable estimate of how the model will perform on unseen data, ensuring that the model generalizes well and does not overfit to a particular subset of the data. In addition, this method utilizes all data points for both training and validation, maximizing the use of available data. Moreover, stratification was employed during this process to maintain the distribution of classes across each fold, thereby preventing biases in model training and evaluation.

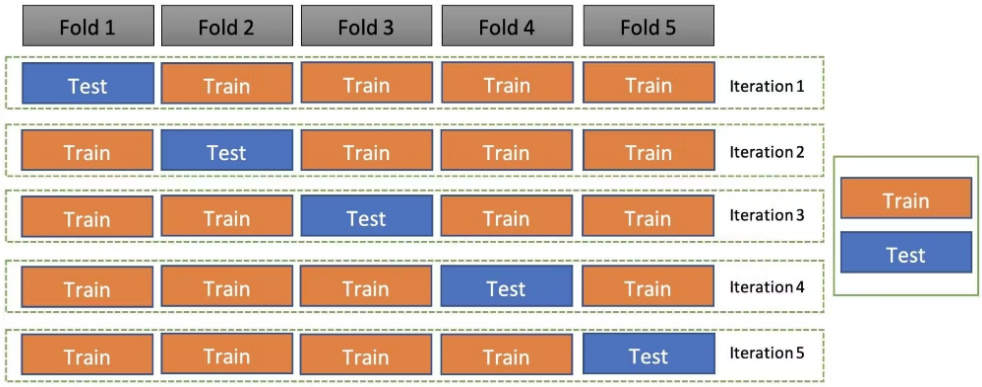

Figure 5.1: Cross-validation explanation [1].

To further enhance the cross-validation setup's robustness, we ensured that rows belonging to the same company were exclusively present in either the training or test set within each fold. This step aimed to prevent data leakage and ensure the model generalized well to unseen company-specific patterns.

5.3 Certainty scores and secondary predictions

An important feature was implemented to improve the utility of these predictions when developing the models to predict the classes of textual data. Beyond predicting the class of the text, we added the certainty scores associated with each prediction and a secondary prediction, which the model considers to be the best prediction after the initial one.

Certainty scores Certainty scores quantify the confidence level of the model in its predictions. These scores are derived from the probability distributions output by the models, indicating how likely the model believes a particular prediction is correct. For instance, a certainty score close to 1 suggests that the model is highly confident in its prediction, whereas a score closer to 0.5 indicates lower confidence.

Each model outputs a probability distribution over all possible classes upon making a prediction. Each class is assigned a certain probability, with the total sum of probabilities being equal to 1. The highest probability in this distribution is taken as the certainty score. For example, if a model predicts a certain class with 85% probability, the certainty score for that prediction is 0.85. These scores are then presented to the end user alongside the predicted class labels.

The inclusion of certainty scores assists the consultants at SINCERIUS in prioritizing their review efforts. Predictions with lower certainty scores are flagged so that the

consultants can judge whether the class was predicted correctly, allowing the user to analyze the ledger account predictions more effectively. This prioritization helps mitigate the risk of misclassification by focusing on cases where the model is less confident.

Secondary predictions In addition to the certainty scores, a secondary prediction is also provided. This secondary prediction represents the model's next best guess for the class label, should the primary prediction be incorrect or uncertain. Including this secondary prediction is helpful in cases where the primary prediction is uncertain or deemed incorrect. The secondary prediction then offers an alternative that is also considered highly probable by the model. This can be particularly useful in edge cases where the model's confidence is distributed among a few classes. This allows the user to see the second most likely classification, which can help make more informed decisions. If the primary prediction has a low certainty score, the user can also consider the secondary prediction as a candidate.

5.4 Hyperparameter tuning

Hyperparameter tuning is the optimization of the model's parameters to improve its performance. Machine learning models often have many adjustable parameters, resulting in various possible configurations associated with a specific performance outcome. To make a fair comparison between traditional machine learning models and BERT, we tuned the hyperparameters of each model to ensure representative performances of the models. Fair comparisons require that each model is given an equal opportunity to perform optimally. Tuning hyperparameters ensures that all models are fine-tuned to their best possible configurations, mitigating the risk of unfair advantages or disadvantages based on arbitrary default settings. This section discusses the hyperparameters we consider for each model.

Logistic regression As discussed in Section 4.1.1, LR models the probability that a given input belongs to a particular class, using a logistic sigmoid function to output values between 0 and 1. Table 5.1 presents the key hyperparameters of the LR model along with their respective value ranges and possible methods.

The 'Max iterations' parameter determines the maximum number of iterations the solver will take before stopping, with a default setting of 100 iterations.[1] Our short text classification problem is complex and possibly needs more iterations to ensure convergence, so we have set a higher upper limit.

[1]All default settings for the Logistic Regression model can be found on the following website: `https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html`

Table 5.1: Hyperparameter Tuning settings for the Logistic Regression model.

Hyperparameter	**Value range/method**
`Max iterations`	$[50, 1500]$
`Penalty`	$[L1, L2]$
`C`	$[5 \times 10^{-3}, 10000]$
`Solver`	$[liblinear, lbfgs, saga]$
`Decision function`	$[OvR, OvO]$

The 'Penalty' parameter considers the regularization technique, which prevents overfitting by adding a penalty term to the loss function. This penalty discourages the model from fitting the noise in the training data, resulting in a generalized model [12]. The L1 penalty adds the absolute values of the coefficients to the loss function, which tends to drive some coefficients to zero, effectively performing feature selection by excluding certain features from the model. The L2 penalty adds the squared values of the coefficients to the loss function. This method distributes the regularization effect across all coefficients, shrinking them towards zero but not making them exactly zero, reducing the variance of the model.

The 'C' parameter is the inverse of regularization strength and is thus linked to the 'Penalty' parameter. A high value for C causes low regularization, and a low value causes high regularization. The default is 1, which results in a moderate influence of the regularization term on the model's coefficients. However, a text classification model often has many features, indicating a strong need for regularization. Therefore, we opted for a rather large possible value range for C.

The 'Solver' parameter determines the optimization algorithm for updating the model's weights in each iteration. The liblinear solver uses a coordinate descent algorithm and is more suitable for smaller datasets [2]. The 'lbfgs' solver stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno. It's a quasi-Newton method that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm and is suited for large datasets [43]. The Stochastic Average Gradient Algorithm (SAGA) is a variant of the stochastic average gradient method and is known for its ability to handle large-scale and sparse datasets. Its primary objective is to minimize the average of many smooth convex functions efficiently [66]. Finally, the multi-label classification strategy is determined by the 'Decision function' parameter. This either selects the OvR or the OvO strategy, which were discussed in Section 4.1.1.

Random Forest The RF model is an ensemble model, making predictions based on the outcome of multiple decision trees (See Section 4.1.2 for more details). The hyperparameters tuning settings for the RF model can be seen in Table 5.2. The 'Criterion'

parameter determines the function used to measure the quality of a split at each node of the trees. The possible settings are 'gini' and 'entropy' (also called information gain), as discussed in Section 4.1.2.

The 'max depth' parameter controls the maximum depth of each decision tree in the forest. Setting a maximum depth for the decision trees helps prevent overfitting, which occurs when the model learns the training data too well and performs poorly on unseen data. A shallow tree (low maximum depth) results in simpler models with fewer splits and may lead to underfitting, while a deep tree (high maximum depth) allows the model to capture more complex relationships in the data and may lead to overfitting.

The 'max features' parameter determines the maximum number of features to consider when looking for the best split at each decision tree node. It is a proportion of the total number of features in the model. This parameter provides flexibility and helps reduce overfitting by limiting the number of features in each tree [11]. The default setting for the 'Max_features' parameter is 'auto', where the maximum number of features equals the square root of the total number of features.[2] Given our extensive feature space, using this value would lead to too many features at each split, resulting in a complex model that will likely overfit. Therefore, we have limited the potential value range to allow for smaller proportions. Moreover, the upper value of 0.1 is approximately equivalent to the default square root setting. Our models typically consist of around 10,000 features (varying depending on the model and dataset), with the square root being 100, representing a proportion of 0.1. Finally, the 'n estimators' parameter controls the number of decision trees n used in the RF ensemble. The default setting is 100, so a range between 50 and 300 seems reasonable.

Table 5.2: Hyperparameter Tuning settings for the Random Forest model.

Hyperparameter	**Value range/method**
`Criterion`	$[gini, entropy]$
`Max depth`	$[50, 15000]$
`Max features`	$[5 \times 10^{-7}, 0.1]$
`n estimators`	$[50, 400]$

Support Vector Machine The hyperparameters of the SVM model are shown in Table 5.3. Again, C is the parameter for the inverse of regularization strength. The range for C was chosen both by trial-and-error and the consideration of a strong regularization influence due to the amount of features in the models, which is why a larger value range was necessary. The 'Kernel' parameter defines the function used to map

[2]All default settings for the Random Forest model can be found on the following website: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.`

the input data into a higher-dimensional space where it becomes easier to classify the data. The kernels considered are the linear kernel, the Radial Basis Function kernel, and the Polynomial kernel, as discussed in Section 4.1.3. Like the LR model, SVM uses the OvR or the OvO technique as a multi-label classification strategy.

Table 5.3: Hyperparameter Tuning settings for the Support Vector Machine model.

Hyperparameter	**Value range/method**
`C`	$[5 \times 10^{-3}, 10000]$
`Kernel`	$[Linear, rbf, poly]$
`Decision function`	$[OvR, OvO]$

BERT There are two main phases in BERT: pre-training and fine-tuning. Hyperparameter optimization in the pre-training phase is possible. However, pre-training involves a dataset of 3.3 billion words and 110 million parameters. This leads to a vast search space, making it highly resource-intensive, requiring significant computational power and time. Each pre-training run can take days to weeks on powerful GPUs or TPUs, and tuning involves running many such experiments, making the process extremely time-consuming [22]. Therefore, The hyperparameter optimization of pre-training is out of the scope of this research. On the other hand, the fine-tuning phase is more suitable for hyperparameter optimization.

Table 5.4 shows the possible hyperparameter values for BERT. The hyperparameter 'Batch size' determines the number of samples from the training dataset utilized during each training step. Predictions are made and compared to the expected outcomes in each step, an error is computed, and the model's internal parameters are adjusted accordingly [75]. A large batch size enables faster computation due to efficient parallel processing but requires more memory and can miss finer details with smoother gradient descent, while a small batch size produces noisier gradient updates aiding in escaping local minima but slows training due to less parallelization and frequent updates [9].

Table 5.4: Hyperparameter Tuning settings for the BERT model.

Hyperparameter	**Value range/method**
`Batch size`	$[16, 32, 64, 128]$
`Epochs`	$[2, 3, 4, 5, 6]$
`Learning rate`	$[5 \times 10^{-7}, 2 \times 10^{-4}]$

An epoch is one complete pass through the entire training dataset. More epochs provide better training opportunities but can lead to overfitting, while fewer epochs reduce the

risk of overfitting but may result in underfitting.

The 'Learning rate' parameter controls the step size at each iteration while moving toward a minimum of the loss function. A high learning rate speeds up the learning process but risks quick convergence to a suboptimal solution or divergence, while a low learning rate ensures more stable convergence but can be slow and may get stuck in local minima [9]. The learning rate is low because of catastrophic forgetting, also known as catastrophic interference. This refers to the neural network's tendency to abruptly forget previously learned information [46]. This causes models like BERT to lose all relevant pre-trained knowledge.

The optimal hyperparameter values are task-specific. However, in the initial study by Devlin et al. [22], the final hyperparameter values were selected using an exhaustive grid search with the following value ranges:

- Batch size: 16, 32
- Number of epochs: 2, 3, 4
- Learning rate: 2×10^{-5}, 3×10^{-5}, 5×10^{-5}

Moreover, they set the dropout probability to 0.1 for all experiments and used the Adam optimizer (short for Adaptive Moment Estimation), an advanced gradient descent optimization algorithm that combines the benefits of two extensions of stochastic gradient descent: the Adaptive Gradient Algorithm and Root Mean Square Propagation. We copied these configurations, except for the hyperparameters in Table 5.4, where the range of potential values has been expanded to increase the chance of discovering the optimal values.

5.5 Optuna optimization

Random and grid search are the most common and straightforward hyperparameter tuning methods. Random search randomly selects hyperparameter values in each iteration and chooses the hyperparameters corresponding to the model's best performance after a set number of iterations. Grid search is a technique where hyperparameters are evaluated for all possible combinations of predefined values. It exhaustively searches the entire hyperparameter space, testing every combination to find the best one according to the specified evaluation metric. Both methods have weaknesses: random search does not cover the entire search space, and grid search is computationally expensive, resulting in a smaller search space to explore.

Therefore, we decided to implement a more sophisticated hyperparameter tuning method called Optuna[3], an open-source framework to automate the hyperparameter optimization for machine learning models.

Objective function The first step in using Optuna is defining an objective function. This function takes the possible hyperparameter values (as discussed in Section 5.4) as input and returns a score that represents the performance of the model trained with those hyperparameters. We decided to use maximizing accuracy as our objective function since the ultimate goal of the model is to make predictions as accurate as possible.

Sampling Algorithm Optuna has several sampling algorithms which can be chosen. Sampling algorithms continuously adjust the search space by assessing the impact of suggested parameter values on the objective value. They converge towards an optimal parameter configuration that maximizes the objective value. The default sampling algorithm of Optuna is the TPESampler (a Tree-structured Parzen Estimator algorithm), which builds a probability distribution model of the objective function based on the observed samples[4]. We have also experimented with the NSGAIISampler (Non-dominated Sorting Genetic Algorithm II), which utilizes an elitist genetic algorithm. However, in our case, the TPESampler demonstrated superior performance. This could be attributed to genetic algorithms often requiring more iterations to converge, and we lacked the computational resources to execute a sufficient number of iterations to reach convergence With the NSGAIISampler.

Pruning algorithms Optuna automatically uses pruning algorithms to terminate unpromising trials during the initial stages of training. This helps save computational

[3] `https://optuna.org/`

[4] See the following website for more details: `https://optuna.readthedocs.io/en/stable/reference/samplers/generated/optuna.samplers.TPESampler.html#optuna.samplers.TPESampler`

resources by stopping trials that are unlikely to yield satisfactory results and consequently accelerates the optimization process. The default pruning algorithm is the MedianPruner, using the median stopping rule, which terminates the trial if its best intermediate result is worse than the median of intermediate results from previous trials at the same step. We have experimented with the recommended SuccessiveHalvingPruner, a bandit-based pruning algorithm. Both pruners resulted in comparable results in all trials.

Trials During optimization, Optuna conducts a series of trials, each corresponding to a set of hyperparameters sampled from the search space. Optuna evaluates the objective function for each trial and uses the results to update its probabilistic model and decide which set of hyperparameters to try next. Since we use 5-fold cross-validation, each trial consists of 5 training and evaluation runs, resulting in a computationally expensive experimental setup. To be able to run sufficient trials for each model, we utilize a cloud GPU environment on Google Colab Pro[5]. We conducted as many number of trials as Google Colab allowed on a T4 GPU environment for each model, leading to varying trial counts across models. Although this might initially suggest an uneven comparison, it ultimately leads to a relatively balanced total runtime across all models.

The results were computed and compared between each model to evaluate their relative performance, and visualizations were made using Neptune[6], an open-source machine learning experiment tracker. All results and graphics are displayed and discussed in Chapter 6.

[5] `https://colab.research.google.com/`
[6] `https://neptune.ai/`

Chapter 6

Results

This chapter presents the results of the classification models applied to the four different datasets. For each dataset, we display the results of the base models, along with the final results after resampling and hyperparameter tuning. We then visualize the impact of the different hyperparameters. Finally, the effect of adding a secondary prediction is evaluated.

6.1 Profit & Loss statement

The Profit & Loss statement dataset is the largest with 54,195 ledger accounts and 11 distinct classes. The classification results of the five models are shown in Table 6.1. It can be seen that BERT outperforms the other models on all performance metrics. Interestingly, BERTje does not outperform BERT, even though the dataset is mostly in Dutch. However, the dataset also includes texts in other languages, such as English and German. While BERT is primarily trained on English texts, it also includes multilingual data that covers various languages, including Dutch and German. This broader coverage may enable BERT to outperform BERTje. It can also be seen that the results only slightly differ from each other for each performance metric. This suggests the models have the ability to correctly classify ledger accounts, both in terms of overall accuracy and in identifying positive instances correctly (precision, recall, and F1-score), striking a good balance between minimizing false positives and false negatives. It also suggests the models are robust and generalize well to unseen data. The F1-scores are lower than both the Precision and Recall scores for most models. This is normally not possible since the F1-score is the harmonic mean of the two (See Equation (5.4)), resulting in the F1-score being positioned between the Precision and Recall scores or being similar to one or both of them (in case they are equal themselves). However, since we use 5-fold cross-validation, we take the average of the results of 5 iterations. If both the Precision and Recall are higher in some iterations, the average F1-score will be lower than the average Precision and Recall scores. This occurs because the F1-score tends to be closer to the lowest score of the two due to the properties of the harmonic mean.

Table 6.1: Cross-validation performances on the Profit & Loss statement.

Model	Accuracy	Precision	Recall	F1-score
LR	0.859	0.863	0.860	0.855
RF	0.838	0.846	0.838	0.836
SVM	0.860	0.864	0.860	0.858
BERTje	0.872	0.878	0.872	0.868
BERT	0.878	0.882	0.878	0.878

Profit & Loss hyperparameter tuning To ensure the models produce representative classification results, hyperparameter tuning is essential, allowing for a fair comparison between the machine learning models. To illustrate the importance of hyperparameter tuning, one significant experiment is discussed for each model. Appendix B shows all other hyperparameter tuning experiments.

Figure 6.1: Profit & Loss hyperparameter experiment with the Logistic Regression model.

Figure 6.1 shows the hyperparameter tuning experiment we did with the LR model, consisting of 200 trials. Each dot in the figure represents a single trial, with the darker blue dots indicating trials conducted towards the end of the experiment. Each trial involves selecting hyperparameter values, training and testing the model for each of the five folds in the cross-validation and then averaging the five accuracies. In total, the model was evaluated 1,000 times. It can be seen that the 'C' hyperparameter shows two hyperbolas, with both peaks lying between 1 and 100. The hyperbola with the lower peak probably corresponds to the 'l1' penalty and the 'lbfgs' and 'saga' solvers. The hyperbola with the higher peak corresponds to the optimal 'l2' penalty hyperparameter and the 'liblinear' solver. The figure shows that choosing the appropriate hyperparameters can increase the objective value (accuracy) by some percentage points, highlighting the importance of hyperparameter tuning.

Figure 6.2 presents the relative hyperparameter importance of the LR model experiment for the Profit & Loss statement. The importance indicates the impact the hyperpa-

rameter has on the objective value. Unfortunately, we could not add the 'penalty' parameter to this figure. The 'lbfgs' only supports the 'l2' penalty, causing errors in the experiment. It can be seen that 'C' is the most crucial hyperparameter, accounting for 67% of the parameters' overall impact on the objective value.

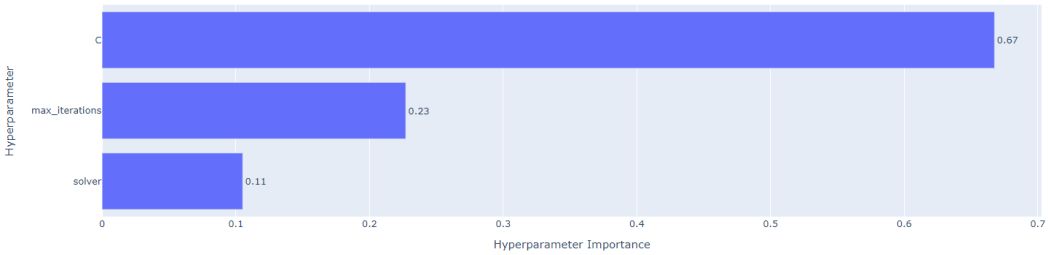

Figure 6.2: Profit & Loss hyperparameter importance of the Logistic Regression model.

The progression of the trials throughout the experiment and their corresponding accuracies are depicted in Figure 6.3. It can be seen that the accuracy is improved by roughly 7% in the first 12 trials, after which it reaches its optimal value. This shows that the search algorithm converged quickly, indicating that the algorithm is effective in identifying the best or near-best solution without requiring extensive computational resources or time.

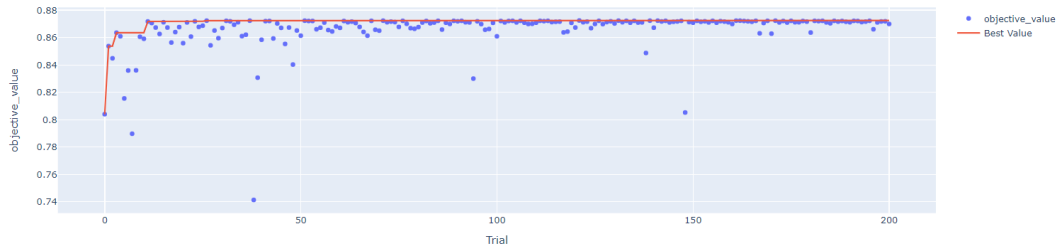

Figure 6.3: Profit & Loss hyperparameter tuning results of the Logistic Regression model.

Final results The final classification results of the five models on the Profit & Loss statement dataset are shown in Figure 6.4. The colored bars indicate the scores without resampling and hyperparameter tuning, and the light blue bars on top of the colored bars show the performance improvement of resampling and hyperparameter tuning. Most of the improvement is attributed to hyperparameter tuning, likely because the class imbalance in the Profit & Loss statement dataset is not significantly present. The RF model gained the most from hyperparameter tuning, whereas the SVM and BERTje models gained the least. Ultimately, the overall comparison between BERT

and the traditional machine learning models remained unchanged after resampling and hyperparameter tuning, with BERT continuing to outperform the other models with an accuracy of 88.9%.

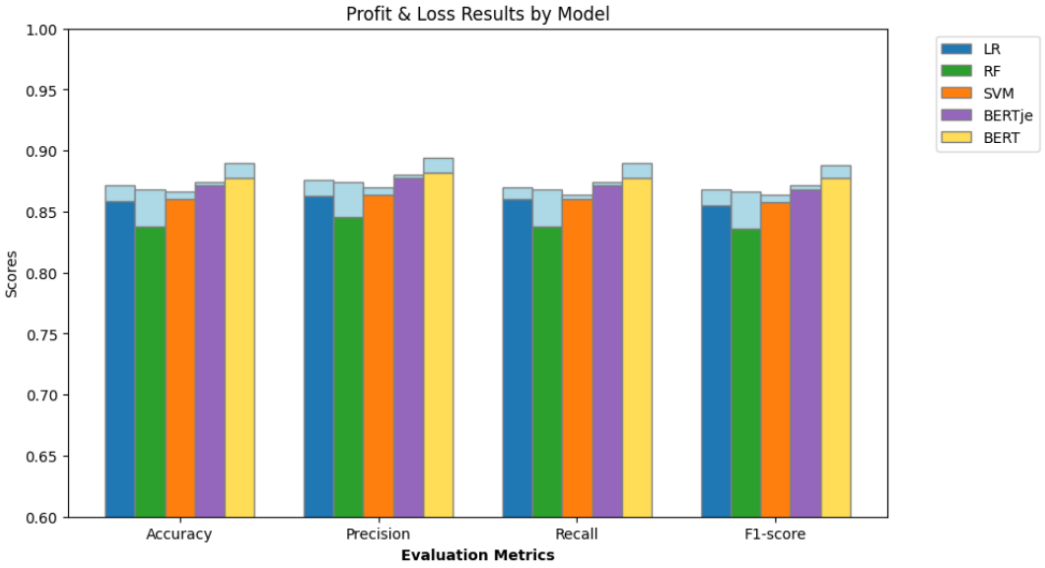

Figure 6.4: Profit & Loss results. The light blue bars indicate the performance improvement after resampling and hyperparameter tuning.

6.2 Balance sheet

The scores of the models for the Balance sheet dataset are depicted in Table 6.2. Again, BERT outperforms the other models on all evaluation metrics. BERTje and RF are the worst-performing models, and LR and SVM are the best traditional machine-learning models. The evaluation scores are closely matched, indicating that the models are also robust for the Balance sheet dataset.

Table 6.2: Cross-validation performances on the Balance sheet.

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
LR	0.828	0.836	0.828	0.828
RF	0.822	0.838	0.820	0.822
SVM	0.827	0.838	0.828	0.828
BERTje	0.822	0.830	0.822	0.820
BERT	0.836	0.842	0.836	0.836

Balance sheet hyperparameter tuning The hyperparameter tuning experiment we did with the RF model for the Balance sheet dataset is shown in Figure 6.5, consisting of 50 trials. The RF model is more computationally intensive than the LR model. Since we conducted the experiments in approximately the same amount of time, the number of trials for the RF model is smaller. The figure visualizes that the 'entropy' criterion slightly outperforms the 'gini' criterion. Setting the 'Max_depth' parameter higher results in better performance. This makes sense since our text data has a high-dimensional feature space. A deeper tree can better navigate this high-dimensional space, identifying the most relevant features for classification. The same goes for the hyperparameter 'n_estimators,' where considering more trees improves the accuracy score. Finally, as discussed in Section 5.4, the default setting for the hyperparameter 'max_features' is set to the square root of the total number of features, resulting in a proportion of roughly 0.01. It can be seen that reducing this proportion increases the final accuracy score by about 2%, further indicating the importance of hyperparameter tuning.

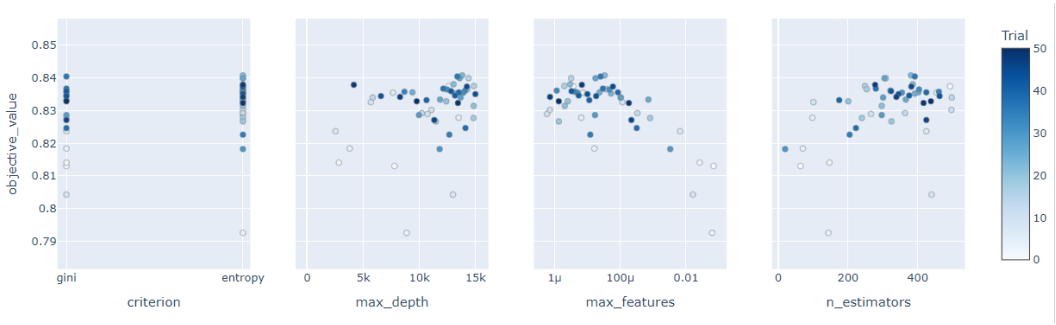

Figure 6.5: Balance sheet hyperparameter experiment with the Random Forest model.

Figure 6.6 presents the relative hyperparameter importance of the RF model experiment for the Balance sheet. It can be seen that the hyperparameter 'max_depth' is most impactful on the objective value, accounting for 69% of the overall impact. The 'max_features' parameter also has some impact, while 'n_estimators' and 'criterion' have minimal influence on the accuracy.

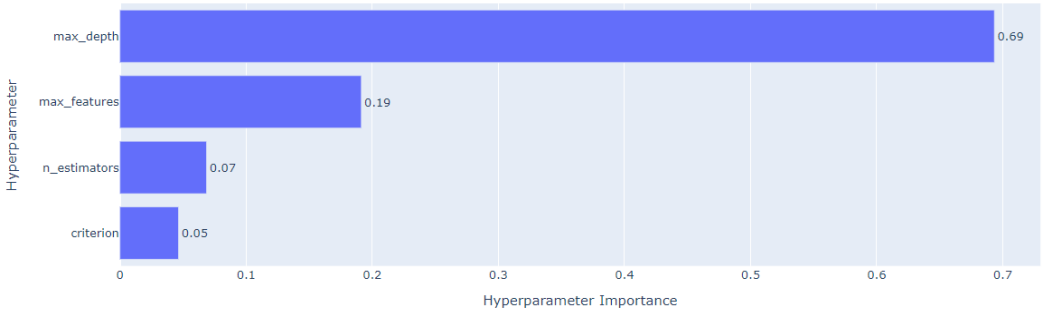

Figure 6.6: Balance sheet hyperparameter importance of the Random Forest model.

The final prediction scores of the models for the Balance sheet dataset are shown in Figure 6.7. Interestingly, after hyperparameter tuning, the BERT and BERTje models have demonstrated only marginal improvements, leading to the RF model surpassing them in performance with an accuracy of 84.1%. This is probably because we could, due to computational resources, only run 6 and 8 trials for the BERT and BERTje models, respectively. The hyperparameter tuning experiments had to be stopped prematurely, resulting in disappointing results. We expect better performances if more trials are run.

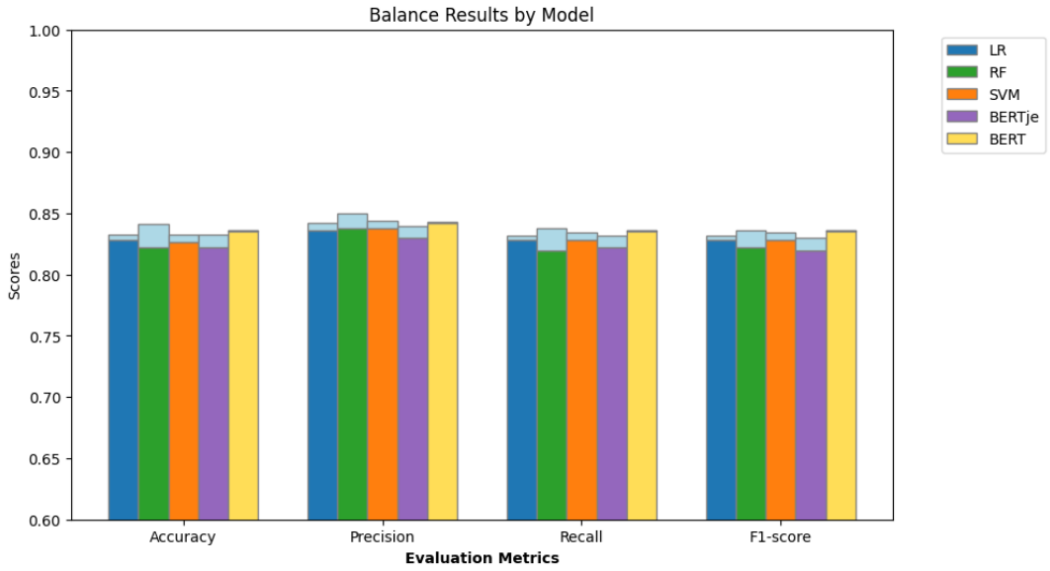

Figure 6.7: Balance sheet final results. The light blue bars indicate the performance improvement after resampling and hyperparameter tuning.

6.3 Personnel expenses

The performance scores of the models for the Personnel Expenses dataset are shown in Table 6.3. We have trained the RoBERTa model for this dataset, which performs

worse than all other models. Moreover, BERT and BERTje are also outperformed by the traditional machine learning models, with the RF model showing the best performance overall. The Personnel Expenses dataset has only four unbalanced classes (See Appendix A.4). This might cause the BERT models to perform worse, suggesting a sensitivity towards unbalanced datasets. Finally, the evaluation scores demonstrate close alignment once more, suggesting that the models exhibit robustness when applied to the Personnel Expenses dataset.

Table 6.3: Cross-validation performances on the personnel expenses dataset.

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
LR	0.802	0.808	0.802	0.800
RF	0.814	0.822	0.814	0.812
SVM	0.808	0.814	0.808	0.806
BERTje	0.784	0.792	0.784	0.778
BERT	0.800	0.802	0.800	0.794
RoBERTa	0.780	0.790	0.780	0.782

Figure 6.8 illustrates the hyperparameter tuning experiment we conducted with the BERT model for the Personnel Expenses dataset, consisting of 20 trials. The 'batch_size' shows optimal values for sizes 64 and 128. The number of 'epochs' does not seem to have much impact on the accuracy score, except that six, and possibly five, epochs result in a decrease in accuracy. This suggests the BERT model tends to overfit quickly when learning the model for too many epochs. Finally, the 'lr' (learning rate) hyperparameter is optimal for a value of 7.06×10^{-5} and experiences catastrophic forgetting (see Section 5.4) for values larger than roughly 1.2×10^{-4}.

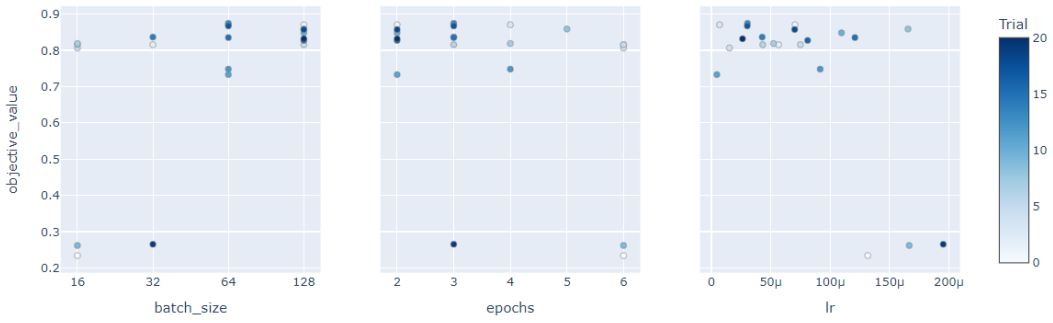

Figure 6.8: Personnel Expenses hyperparameter experiment with the BERT model.

The hyperparameter importance of the BERT model for the Personnel Expenses dataset is depicted in Figure 6.9. Most of the hyperparameter's impact on maximizing the accuracy comes from the 'lr' hyperparameter, with a relative importance of 0.85.

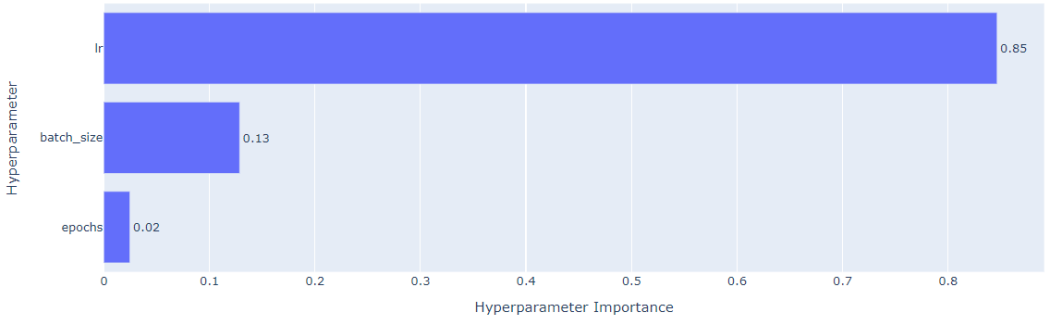

Figure 6.9: Personnel Expenses hyperparameter importance of the BERT model.

Figure 6.10 shows the final classification scores of the models for the Personnel Expenses dataset. The red bars of the RF model results indicate a decrease in performance, showing that resampling can potentially lead to worse performance. This can be caused by loss of information in our undersampling step or by introducing noise in our oversampling SMOTE technique. Interestingly, the BERT and BERTje models have shown significant performance increases, surpassing the traditional machine learning models. This contrasts the earlier findings before resampling and hyperparameter tuning were applied. This demonstrates handling data imbalance in combination with hyperparameter tuning is crucial for the BERT and BERTje models.

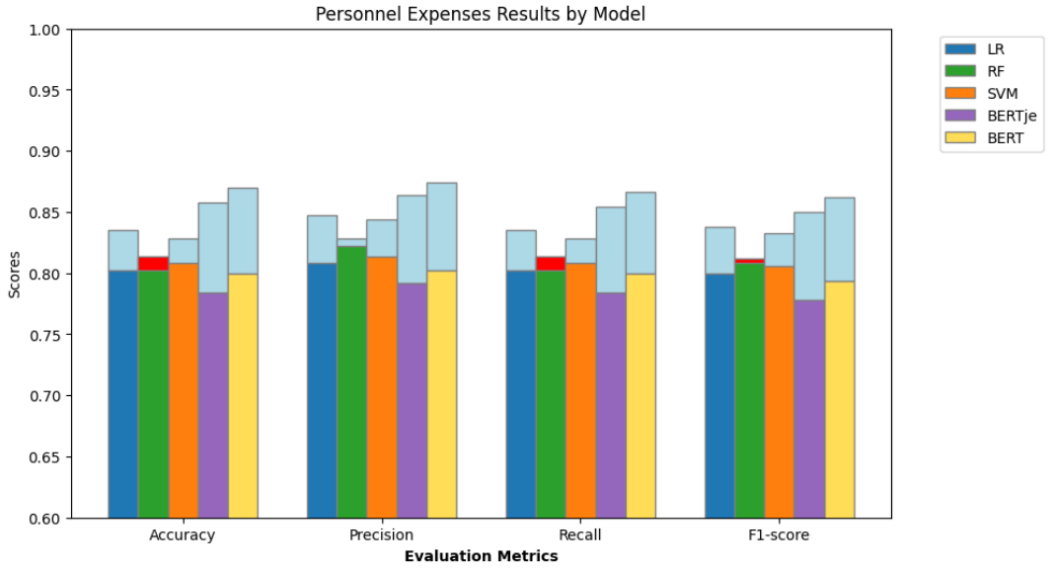

Figure 6.10: Personnel Expenses final results. The light blue bars indicate the performance improvement after resampling and hyperparameter tuning.

6.4 Operational expenses

Table 6.4 shows the performance scores of the models for the Operational Expenses dataset. We have trained the DistilBERT model for this dataset, which performs worse than all other models. BERTje outperforms BERT for this dataset. The Operational Expenses dataset contains relatively more Dutch pieces of text, probably resulting in BERTje learning the patterns in the training data better. BERT is also outperformed by the RF model, which shows the best performance of the traditional machine learning models. In conclusion, the evaluation scores once again indicate close alignment, implying that the models maintain robustness when applied to the Operational Expenses dataset.

Table 6.4: Cross-validation performances on the operational expenses dataset.

Model	Accuracy	Precision	Recall	F1-score
LR	0.697	0.684	0.698	0.668
RF	0.710	0.704	0.710	0.682
SVM	0.690	0.680	0.690	0.660
BERTje	0.722	0.724	0.722	0.698
BERT	0.700	0.674	0.700	0.674
DistilBERT	0.662	0.560	0.662	0.588

Our conducted experiment with the SVM model on the Operational Expenses dataset, consisting of 100 trials, is depicted in Figure 6.11. Similar to the LR experiment, the 'C' hyperparameter exhibits hyperbolas. Three distinct hyperbolas are visible, likely representing one for each kernel. The 'poly' kernel exhibits the lowest peak topping just above 60% accuracy, the 'rbf' kernel falls in the middle range, and the 'linear' kernel yields the highest achievable accuracies just above 70%. The 'OvR' decision function slightly outperformed the 'OvO' function for the SVM model for the Operational Expenses dataset.

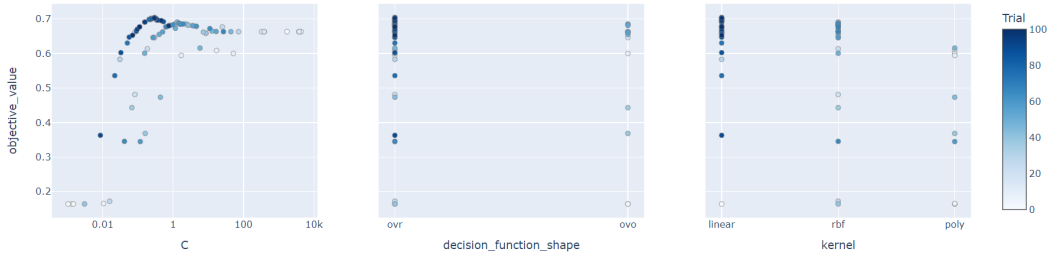

Figure 6.11: Operational Expenses hyperparameter experiment with the SVM model.

Interestingly, towards the end of the experiment (the darker blue dots), the search algorithm mainly explores the more promising hyperparameter values. This suggests that it has found a local (or possibly global) maximum and is attempting to fine-tune the values further to reach the actual maximum in this search area.

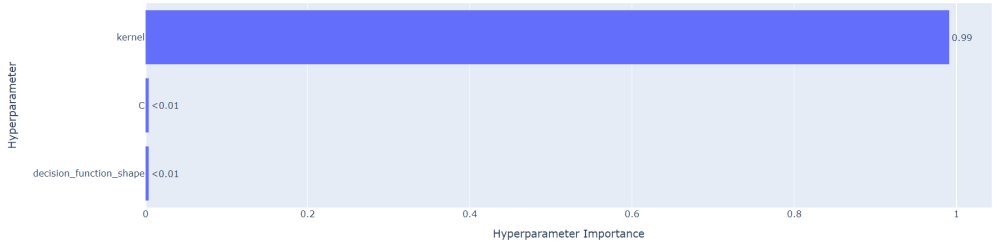

Figure 6.12: Operational Expenses hyperparameter importance of the SVM model.

The 'kernel' hyperparameter clearly has the most impact on maximizing the accuracy, as shown in Figure 6.12. However, we should note that the importance of the parameter 'C' is somewhat distorted due to the three overlapping hyperbolas and are, in fact, having quite a significant impact, as we can see in Figure 6.11.

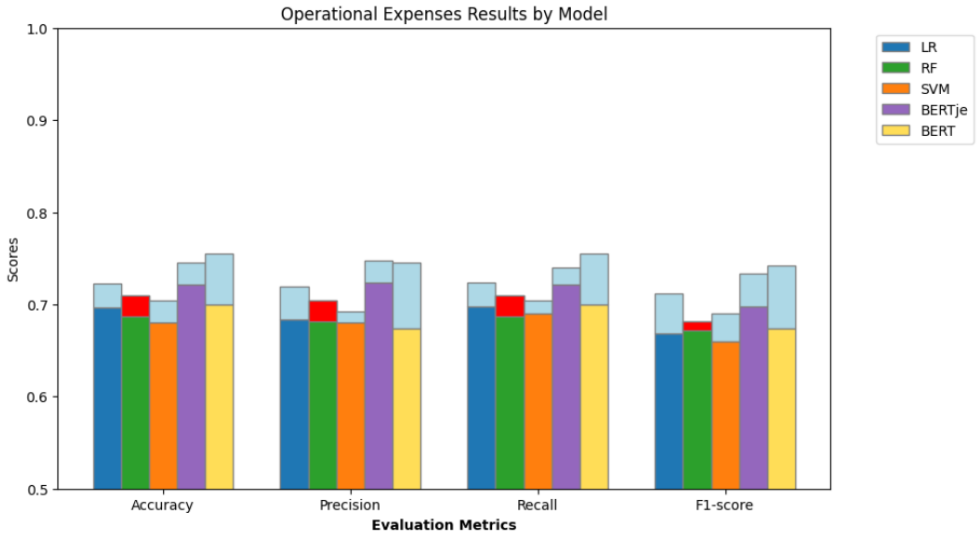

Figure 6.13: Operational Expenses final results. The light blue bars indicate the performance improvement after resampling and hyperparameter tuning.

The final results of the Operational Expenses dataset are shown in Figure 6.13. In line with the final results for the Personnel Expenses, the RF model computed worse performance after resampling and hyperparameter tuning. Furthermore, the BERT model

has improved significantly again, outperforming the other models in all evaluation metrics except for precision, where BERTje takes the lead.

6.5 Optimal hyperparameters

This section examines the best-performing hyperparameters that were found for each model across the different datasets. The corresponding visualizations of the hyperparameter tuning experiments are shown in Appendix B.

The optimal hyperparameter values for the LR model are depicted in Table 6.5. The 'Liblinear' solver results in the best performance across all datasets. The maximum number of iterations varies somewhat, with the value for the Profit & Loss statement being significantly lower. This can probably be assigned to the model overfitting, which could happen due to noisy data in this dataset. The 'C' parameter is high for the Profit & Loss dataset, moderate for the Balance sheet dataset (close to 1), and low for the Personnel and Operational Expenses datasets. Since C is the inverse of regularization strength, this suggests that it is optimal to use most of the features of the Profit & Loss dataset and penalize using many features in the Expenses datasets. Finally, the optimal penalty is 'l2', and the optimal decision function is 'OvR' for all datasets.

Table 6.5: Optimal hyperparameters for each dataset for the Logistic Regression model.

Hyperparameter	**P&L statement**	**Balance sheet**	**Personnel expenses**	**Operational expenses**
Solver	Liblinear	Liblinear	Liblinear	Liblinear
Max iterations	403	1066	891	986
C	19.12	1.28	0.027	0.573
Penalty	l2	l2	l2	l2
Decision function	OvR	OvR	OvR	OvR

Table 6.6 shows the optimal hyperparameters of the RF model for all datasets. The 'N estimators' and 'Criterion' hyperparameters did not significantly impact the accuracy (see Figure 6.6). They do vary, with the Operational Expenses dataset having a lower optimal number of iterations and the Balance sheet dataset having the 'Entropy' criterion as the optimal setting instead of 'Gini'. The optimal maximum depth of the trees is fairly consistent across all datasets, as was expected, since a large maximum depth is preferred, as was discussed in Section 6.2. The optimal maximum proportion of the features varies a lot, which is not surprising since the total number of features differs greatly across the different datasets.

Table 6.6: Optimal hyperparameters for each dataset for the Random Forest model.

Hyperparameter	**P&L statement**	**Balance sheet**	**Personnel expenses**	**Operational expenses**
N estimators	241	381	429	158
Criterion	Gini	Entropy	Gini	Gini
Max depth	12,622	13,812	11,893	12,832
Max features	2.11×10^{-6}	3.37×10^{-5}	8.00×10^{-4}	1.85×10^{-8}

The SVM model's optimal hyperparameter values can be seen in Table 6.7. The optimal values for 'C' are similar to those of the LR model, with higher values for larger datasets (P&L statement and Balance sheet) and lower values for the smaller ones. Furthermore, the optimal kernel for larger datasets is 'rbf', while for the smaller datasets it is 'linear'. Finally, the optimal decision function is OvR again, except for the Profit & Loss dataset. However, the decision function does not significantly impact the accuracy, showing only marginal differences between the two (see Figure 6.12).

Table 6.7: Optimal hyperparameters for each dataset for the Support Vector Machine model.

Hyperparameter	**P&L statement**	**Balance sheet**	**Personnel expenses**	**Operational expenses**
C	4.10	4.26	0.040	0.299
Kernel	rbf	rbf	Linear	Linear
Decision function	OvO	OvR	OvR	OvR

Table 6.8 shows the optimal hyperparameter values for the BERTje model. The learning rate, which is the most impactful hyperparameter, varies between 1.88×10^{-5} and 7.42×10^{-5}, which is not surprising, considering the optimal values found in the study of Devlin et al. [22], which ranges from 2×10^{-5} to 5×10^{-5}. The optimal batch sizes differ for each dataset, indicating varying dataset complexities. These differences could affect the model's learning dynamics and the efficiency of gradient updates during training. Finally, the optimal number of epochs is three for the larger datasets and four and five for the Operational and Personnel Expenses datasets, respectively.

The optimal hyperparameter values for the BERT model are displayed in Table 6.9. The optimal learning rate values are somewhat unexpected, as they are notably larger, with the learning rate of the Operational Expenses dataset of 1.00×10^{-4} even approaching the catastrophic forgetting levels (see Section 6.3). The optimal batch sizes present a distinct pattern compared to those observed for BERTje. This could stem

Table 6.8: Optimal hyperparameters for each dataset for the BERTje model.

Hyperparameter	P&L	Balance	Personnel ex.	Operational ex.
Learning rate	3.05×10^{-5}	7.42×10^{-5}	1.88×10^{-5}	2.07×10^{-5}
Batch size	64	128	32	16
Epochs	3	3	5	4

from the different pre-training approaches utilized in the models. However, it could also be attributed to insufficient computational resources to achieve convergence in the tuning experiments. In conclusion, the optimal number of epochs differs across all datasets. However, this hyperparameter is less noteworthy as it has minimal impact on the accuracy, except for some instances with 6 epochs where the model displayed slight overfitting.

Table 6.9: Optimal hyperparameters for each dataset for the BERT model.

Hyperparameter	P&L	Balance	Personnel ex.	Operational ex.
Learning rate	6.81×10^{-5}	6.72×10^{-5}	7.06×10^{-5}	1.00×10^{-4}
Batch size	16	16	128	128
Epochs	4	5	2	3

6.6 Secondary prediction

In this section, we evaluate the impact of adding a secondary prediction on determining the right class of the ledger accounts. We are only interested in evaluating the accuracy of the models since their robustness has already been tested. Pure classification performance is our main focus now. As mentioned in Section 5.3, the predictions are based on the certainty scores, where the model classifies the ledger accounts based on the class with the highest certainty. The secondary prediction is made by simply selecting the class with the second-highest certainty score.

Figure 6.14 shows the accuracies of the models, including the total accuracy of considering both the primary and secondary predictions. The colored bars indicate the final accuracies we have found through the hyperparameter tuning. The light green bars highlight the combined accuracy, where predictions are considered correct in case either of the two contains the right classification. All accuracies have improved significantly beyond what would be achieved by randomly selecting a secondary prediction. This ultimately helps the end-users determine the right ledger accounts classes, know-

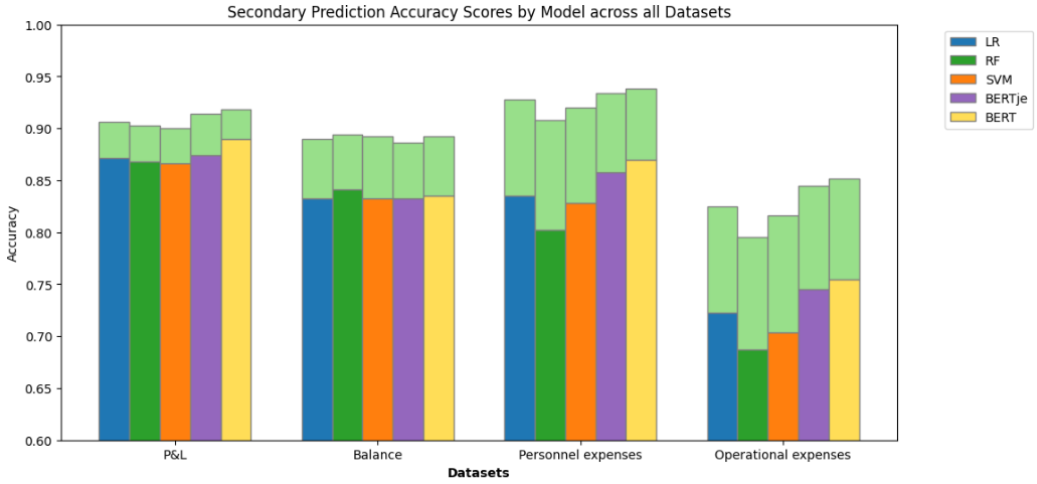

Figure 6.14: Final model accuracies and secondary predictions (indicated by the light green bars).

ing that the correct class is one of the two in roughly 90% of the cases.

The Profit & Loss and Balance datasets improved less than the Personnel and Operational Expenses datasets. This makes sense because the latter two contain only four and six distinct classes, making it more likely the second prediction is the correct one. In conclusion, the BERT model performs better than the other models across all datasets except for the Balance sheet, where accuracies are comparable, and the RF model achieves the highest accuracy.

Chapter 7

Conclusions

To conclude, we have compared the traditional machine learning models, Logistic Regression, Random Forest, and Support Vector Machine, with the state-of-the-art Natural Language Processing model BERT and its Dutch variant BERTje for the task of short financial text classification. The data for this comparison was provided by SINCERIUS, a financial due diligence company. It consists of general ledgers from 215 companies, each containing multiple accounts. Such an account is essentially a short financial description that groups all related transactions. General ledger accounts are part of one of the two primary financial statements: the balance sheet or the Profit and Loss statement. Moreover, the personnel and operational expenses classes on the profit and loss statement can be further divided into subclasses, resulting in four distinct datasets. Our objective was to evaluate the models' performance across the four datasets. We evaluated performance based on four metrics: accuracy, precision, recall, and the F1-score. Additionally, we have tuned the hyperparameters of all models across all datasets to ensure fair comparisons could be made through representative classification scores. This chapter concludes with several recommendations for further research.

7.1 Main results

The results of our experiments highlight the strengths and limitations of each approach. All models showed relatively small differences across all four evaluation metrics, which implies the models are robust and generalize well to unseen data. BERT outperformed the traditional models in classifying financial texts related to the balance sheet and the Profit & Loss statement. BERTje showed the best classification results for the operational expenses dataset, while the Random Forest model achieved the best performance on the personnel expenses dataset. These varying results indicate that the classification performance of machine learning models depends on the specific characteristics and complexity of the datasets, preventing us from drawing any conclusions on whether the pre-trained BERT model consistently outperforms traditional machine

learning models. Notably, BERT performed better on the two larger datasets than the other models, suggesting that its ability to leverage large amounts of data and capture nuanced contextual information may provide an advantage when dealing with more extensive datasets.

7.2 Hyperparameter tuning

Different hyperparameter settings can significantly affect a model's performance. By tuning these parameters, we aimed to find the settings that yield the best performance on the specific datasets. Models with default hyperparameters may not be optimized for the specific characteristics of the dataset being used. This can lead to biased results where one model appears to perform better simply because its default settings happen to be more suitable for the dataset. Thus, tuning hyperparameters ensures fair comparisons between the models.

After hyperparameter tuning, BERT exhibited the highest performance across all datasets except the Balance sheet, where Random Forest slightly outperformed it. This discrepancy suggests that BERT's hyperparameters may not have been optimally tuned, given the limited scope of our tuning experiment (only 6 trials were conducted on the Balance sheet dataset). Nonetheless, the tuned BERT model demonstrates strong potential for generalizability.

In conclusion, our experiments show that BERT does not consistently outperform the traditional machine learning models, showing superior performance only for larger datasets. However, when proper data imbalance handling and hyperparameter tuning are executed, BERT tends to be superior, even for smaller datasets. Therefore, to address our research question regarding the comparative performance of traditional machine learning models versus the state-of-the-art BERT model in classifying short financial texts, BERT demonstrates superior classification capabilities, provided appropriate resampling and hyperparameter tuning techniques are implemented.

It is worth noting that training BERT requires significantly more time than training traditional machine learning models. Given this, along with the substantial time needed for hyperparameter tuning, one must evaluate whether the computational investment to train advanced embedding models like BERT outweighs the performance gains over traditional machine learning models.

7.3 Further research

Recently, hybrid models have gained interest in the literature, showing promising results. Hybrid approaches aim to combine BERT with other machine learning approaches to enhance performance. One such strategy is to add Long Short-Term Memory (LSTM)

or Gated Recurrent units (GRU) layers on top of BERT embeddings to capture additional sequential dependencies in the data [50, 73]. These studies are still in their early stages and have not yet surpassed the base BERT model in performance. However, they have demonstrated faster processing speeds, facilitating effective hyperparameter tuning. It is anticipated that hybrid BERT models will outperform BERT in the near future, particularly with extended training over more epochs [73].

Another direction could be to build ensemble models, combining the strengths of traditional machine learning methods and pre-trained models such as BERT. The concept of ensemble models involves combining the predictions of multiple individual models to make a final prediction. This can be done by either taking the majority vote of the individual models' predictions or by averaging their certainty scores and selecting the class with the highest score. This approach is similar to the Random Forest method, which is an ensemble technique that uses multiple decision trees to make predictions. Despite its simplicity, ensembling BERT with other machine learning models [4] or with other BERT variants [14, 18] has shown promising results.

The Google Colab environment limits the usage duration of the powerful T4 GPU, forcing us to terminate the hyperparameter tuning process of the BERT and BERTje models prematurely. Consequently, this possibly led to sub-optimal performances of the BERT models. We could conduct only several trials, whereas running dozens of trials would have been preferable for thorough hyperparameter optimization. Future research should consider utilizing environments with more extensive computational resources to fully explore the potential of hyperparameter tuning for BERT models, ensuring optimal performance.

Finally, we would recommend that SINCERIUS explore further fine-tuning the models with a focus on accuracy, including secondary predictions. To achieve this, the objective function of hyperparameter tuning experiments should prioritize maximizing the combined accuracy of both primary and secondary predictions. Improving accuracy in this manner would help consultants make quick and reliable decisions about classifying ledger accounts.

Bibliography

[1] Different Types of Cross-Validations in Machine Learning. 2022. URL: https://www.turing.com/kb/different-types-of-cross-validations-in-machine-learning-and-their-explanations (visited on 04/24/2023).

[2] Scikit learn. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (visited on 05/15/2024).

[3] Basant Agarwal and Namita Mittal. "Text classification using machine learning methods-a survey". In: *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), December 28-30, 2012*. Springer. 2014, pp. 701–709.

[4] Ionuţ-Alexandru Albu and Stelian Spınu. "Emotion detection from tweets using a BERT and SVM ensemble model". In: *arXiv preprint arXiv:2208.04547* (2022).

[5] Daler Ali, Malik Muhammad Saad Missen, and Mujtaba Husnain. "Multiclass event classification from text". In: *Scientific Programming* 2021 (2021), pp. 1–15.

[6] Yusuf Arslan et al. "A comparison of pre-trained language models for multi-class text classification in the financial domain". In: *Companion Proceedings of the Web Conference 2021*. 2021, pp. 260–268.

[7] Chiara Bardelli et al. *Automatic electronic invoice classification using machine learning models*. SOCIAL SCIENCES, Oct. 5, 2020. DOI: 10.20944/preprints20.2010.0057.v1.

[8] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. "A neural probabilistic language model". In: *Advances in neural information processing systems* 13 (2000).

[9] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA, 2017.

[10] Hampus Bengtsson and Johannes Jansson. "Using classification algorithms for smart suggestions in accounting systems". In: (2015).

[11] Simon Bernard, Laurent Heutte, and Sébastien Adam. "Influence of hyperparameters on random forest accuracy". In: *Multiple Classifier Systems: 8th International Workshop, MCS 2009, Reykjavik, Iceland, June 10-12, 2009. Proceedings 8*. Springer. 2009, pp. 171–180.

[12] Christopher M Bishop. "Pattern recognition and machine learning". In: *Springer* 2 (2006), pp. 1122–1128.

[13] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[14] J Briskilal and CN Subalalitha. "An ensemble model for classifying idioms and literal texts using BERT and RoBERTa". In: *Information Processing & Management* 59.1 (2022), p. 102756.

[15] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[16] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[17] Andrew M Dai and Quoc V Le. "Semi-supervised sequence learning". In: *Advances in neural information processing systems* 28 (2015).

[18] Huong Dang et al. "Ensemble BERT for classifying medication-mentioning tweets". In: *Proceedings of the Fifth Social Media Mining for Health Applications Workshop & Shared Task*. 2020, pp. 37–41.

[19] Wietse De Vries et al. "Bertje: A dutch bert model". In: *arXiv:1912.09582* (2019).

[20] Scott Deerwester et al. "Indexing by latent semantic analysis". In: *Journal of the American society for information science* 41.6 (1990), pp. 391–407.

[21] Guillaume Desagulier. *Word embeddings: The (very) basics*. Apr. 2018. URL: `https://corpling.hypotheses.org/495`.

[22] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[23] *EU rules on financial information disclosed by companies*. Accessed: 05-02-2024. (2023). URL: `https://finance.ec.europa.eu/capital-markets-union-and-financial-markets/company-reporting-and-auditing/company-reporting/financial-reporting_en`.

[24] Mengzhen Fan et al. "Fusing global domain information and local semantic information to classify financial documents". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 2413–2420.

[25] Hossam Faris, Ibrahim Aljarah, and Seyedali Mirjalili. "Training feedforward neural networks using multi-verse optimizer for binary classification problems". In: *Applied Intelligence* 45 (2016), pp. 322–332.

[26] Wissal Farsal, Samir Anter, and Mohammed Ramdani. "Deep learning: An overview". In: *Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications*. 2018, pp. 1–6.

[27] Christopher A Flores and Rodrigo Verschae. "A generic semi-supervised and active learning framework for biomedical text classification". In: *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE. 2022, pp. 4445–4448.

[28] Silvia Garcia-Mendez et al. "Identifying banking transaction descriptions via support vector machine short-text classification based on a specialized labelled corpus". In: *IEEE Access* 8 (2020), pp. 61642–61655.

[29] Andrea Gasparetto et al. "A survey on text classification algorithms: From text to predictions". In: *Information* 13.2 (2022), p. 83.

[30] Santiago González-Carvajal and Eduardo C Garrido-Merchán. "Comparing BERT against traditional machine learning text classification". In: *arXiv:2005.13012* (2020).

[31] Venu G Gudise and Ganesh K Venayagamoorthy. "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks". In: *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*. IEEE. 2003, pp. 110–117.

[32] Xu Han et al. "Pre-trained models: Past, present and future". In: *AI Open* 2 (2021), pp. 225–250.

[33] Feras Al-Hawari and Hala Barham. "A machine learning based help desk system for IT service management". In: *Journal of King Saud University-Computer and Information Sciences* 33.6 (2021), pp. 702–718.

[34] Jeremy Howard and Sebastian Ruder. "Universal language model fine-tuning for text classification". In: *arXiv preprint arXiv:1801.06146* (2018).

[35] Prabhjot Kaur. "Web content classification: A survey". In: *arXiv:1405.0580* (2014).

[36] S. Sathiya Keerthi et al. "Improvements to Platt's SMO algorithm for SVM classifier design". In: *Neural computation* 13.3 (2001), pp. 637–649.

[37] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[38] Will Koehrsen. *Random Forest Simple Explanation*. Dec. 2017. URL: `https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d`.

[39] Anis Koubaa. "GPT-4 vs. GPT-3.5: A Concise Showdown". In: *Preprints* (Mar. 2023). DOI: `10.20944/preprints202303.0422.v1`. URL: `https://doi.org/10.20944/preprints202303.0422.v1`.

[40] Kamran Kowsari et al. "Hdltex: Hierarchical deep learning for text classification". In: *2017 16th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2017, pp. 364–371.

[41] David D Lewis. "An evaluation of phrasal and clustered representations on a text categorization task". In: *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*. 1992, pp. 37–50.

[42] Xiaoli Li and Bing Liu. "Learning to classify texts using positive and unlabeled data". In: *IJCAI*. Vol. 3. 2003. Citeseer. 2003, pp. 587–592.

[43] Dong C Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[44] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[45] Sameen Maruf, Kashif Javed, and Haroon A Babri. "Improving text classification performance with random forests-based feature selection". In: *Arabian Journal for Science and Engineering* 41 (2016), pp. 951–964.

[46] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[47] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems* 26 (2013).

[48] Shervin Minaee et al. "Deep learning–based text classification: a comprehensive review". In: *ACM computing surveys (CSUR)* 54.3 (2021), pp. 1–40.

[49] Tetsuji Nakagawa, Kentaro Inui, and Sadao Kurohashi. "Dependency tree-based sentiment classification using CRFs with hidden variables". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. 2010, pp. 786–794.

[50] K Nithya et al. "Hybrid approach of deep feature extraction using BERT–OPCNN & FIAC with customized Bi-LSTM for rumor text classification". In: *Alexandria Engineering Journal* 90 (2024), pp. 65–75.

[51] Juho Ojala. "Machine learning in automating bank statement postings". In: 2018. URL: `https://api.semanticscholar.org/CorpusID:69320475`.

[52] Cristian Padurariu and Mihaela Elena Breaban. "Dealing with data imbalance in text classification". In: *Procedia Computer Science* 159 (2019), pp. 736–745.

[53] Amichai Painsky and Gregory Wornell. "On the universality of the logistic loss function". In: *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2018, pp. 936–940.

[54] Upasana Pandey and Shampa Chakravarty. "A survey on text classification techniques for e-mail filtering". In: *2010 Second International Conference on Machine Learning and Computing*. IEEE. 2010, pp. 32–36.

[55] Nikiforos Pittaras et al. "Text classification with semantically enriched word embeddings". In: *Natural Language Engineering* 27.4 (2021), pp. 391–425.

[56] Matt Post and Shane Bergsma. "Explicit and implicit syntactic features for text classification". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2013, pp. 866–872.

[57] Rukhma Qasim et al. "A fine-tuned BERT-based transfer learning approach for text classification". In: *Journal of healthcare engineering* 2022 (2022).

[58] Colin Raffel et al. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[59] Irina Raicu, Răzvan Bologa, and Radu Constantinescu. "Multi-class text supervised classification on Romanian financial banking reviews". In: *Proceedings of the 18th International Conference on Informatics in ECONOMY Education, Research and Business Technologies*. 2019, pp. 31–36.

[60] Chinmayi Ramasubramanian and R Ramya. "Effective pre-processing activities in text mining using improved porter's stemming algorithm". In: *International Journal of Advanced Research in Computer and Communication Engineering* 2.12 (2013), pp. 4536–4538.

[61] Muhammad Raza et al. "A comparative analysis of machine learning models for quality pillar assessment of SaaS services by multi-class text classification of users' reviews". In: *Future generation computer systems* 101 (2019), pp. 341–371.

[62] Ben Rodrawangpai and Witawat Daungjaiboon. "Improving text classification with transformers and layer normalization". In: *Machine Learning with Applications* 10 (2022), p. 100403.

[63] Anshul Saini. *Guide on Support Vector Machine (SVM) Algorithm*. Jan. 2024. URL: `https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/`.

[64] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108* (2019).

[65] Justyna Sarzynska-Wawer et al. "Detecting formal thought disorder by deep contextualized word representations". In: *Psychiatry Research* 304 (2021), p. 114135.

[66] Mark Schmidt, Nicolas Le Roux, and Francis Bach. "Minimizing finite sums with the stochastic average gradient". In: *Mathematical Programming* 162 (2017), pp. 83–112.

[67] Sebastian Schmidt, Steffen Schnitzer, and Christoph Rensing. "Text classification based filters for a domain-specific search engine". In: *Computers in Industry* 78 (2016), pp. 70–79.

[68] Fabrizio Sebastiani. "Machine learning in automated text categorization". In: *ACM computing surveys (CSUR)* 34.1 (2002), pp. 1–47.

[69] S Selva Birunda and R Kanniga Devi. "A review on word embedding techniques for text classification". In: *Innovative Data Communication Technologies and Application: Proceedings of ICIDCA 2020* (2021), pp. 267–281.

[70] Kanish Shah et al. "A comparative analysis of logistic regression, random forest and KNN models for the text classification". In: *Augmented Human Research* 5.1 (2020), p. 12.

[71] Yijun Shao et al. "Clinical text classification with word embedding features vs. bag-of-words features". In: *2018 IEEE International conference on big data (big data)*. IEEE. 2018, pp. 2874–2878.

[72] Hongbo Shi et al. "A hybrid sampling method based on safe screening for imbalanced datasets with sparse structure". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[73] S Shreyashree et al. "BERT-based hybrid RNN model for multi-class text classification to study the effect of pre-trained word embeddings". In: *International Journal of Advanced Computer Science and Applications* 13.9 (2022).

[74] Sara Silva, Rúben Pereira, and Ricardo Ribeiro. "Machine learning in incident categorization automation". In: *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE. 2018, pp. 1–6.

[75] Rafael Silva Barbon and Ademar Takeo Akabane. "Towards Transfer Learning Techniques—BERT, DistilBERT, BERTimbau, and DistilBERTimbau for Automatic Text Classification from Different Languages: A Case Study". In: *Sensors* 22.21 (2022), p. 8184.

[76] Jasmeet Singh and Vishal Gupta. "Text stemming: Approaches, applications, and challenges". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), pp. 1–46.

[77] Jia Song et al. "A bi-directional sampling based on K-means method for imbalance text classification". In: *2016 IEEE/ACIS 15th international conference on computer and information science (ICIS)*. IEEE. 2016, pp. 1–5.

[78] Chi Sun, Luyao Huang, and Xipeng Qiu. "Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence". In: *arXiv:1903.09588* (2019).

[79] Shan Suthaharan and Shan Suthaharan. "Support vector machine". In: *Machine learning models and algorithms for big data classification: thinking with examples for effective learning* (2016), pp. 207–235.

[80] T. Vink, P. V. D. Putten, and Erik Van De Ven. "Master Computer Science Classification of bank transactions into multi-class, non-uniformly distributed ledger accounts". In: (2022).

[81] Ayisha Tabassum and Rajendra R Patil. "A survey on text pre-processing & feature extraction techniques in natural language processing". In: *International Research Journal of Engineering and Technology (IRJET)* 7.06 (2020), pp. 4864–4867.

[82] Adil Yaseen Taha et al. "Multilabel over-sampling and under-sampling with class alignment for imbalanced multilabel text classification". In: *Journal of Information and Communication Technology* 20.3 (2021), pp. 423–456.

[83] Wilson L Taylor. ""Cloze procedure": A new tool for measuring readability". In: *Journalism quarterly* 30.4 (1953), pp. 415–433.

[84] Simon Tong and Daphne Koller. "Support vector machine active learning with applications to text classification". In: *Journal of machine learning research* 2.Nov (2001), pp. 45–66.

[85] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[86] Yasmen Wahba, Nazim Madhavji, and John Steinbacher. "A comparison of svm against pre-trained language models (plms) for text classification tasks". In: *International Conference on Machine Learning, Optimization, and Data Science*. Springer. 2022, pp. 304–313.

[87] Hao Wang and Sanhong Deng. "A paper-text perspective: Studies on the influence of feature granularity for Chinese short-text-classification in the Big Data era". In: *The Electronic Library* 35.4 (2017), pp. 689–708.

[88] Sida I Wang and Christopher D Manning. "Baselines and bigrams: Simple, good sentiment and topic classification". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2012, pp. 90–94.

[89] Wei Wang et al. "Financial Numeral Classification Model Based on BERT". In: *NII Testbeds and Community for Information Access Research*. Ed. by Makoto P. Kato et al. Vol. 11966. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 193–204. DOI: 10.1007/978-3-030-36805-0_15. URL: https://doi.org/10.1007/978-3-030-36805-0_15.

[90] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144* (2016).

[91] Wei Yin and Liping Shen. "A short text classification approach with event detection and conceptual information". In: *Proceedings of the 2020 5th International Conference on Machine Learning Technologies*. 2020, pp. 129–135.

[92] Qianhao Yu and Xuhui Zhang. "Application of Data Augmentation in Financial Sentiment Analysis Task". In: *Available at SSRN 3969563* (2021).

[93] Wenjie Zhao et al. "The study on the text classification for financial news based on partial information". In: *IEEE Access* 8 (2020), pp. 100426–100437.

Appendix A

Ledger account classes

Table A.1: Classes balance sheet

Class	**Frequency**
Tax	6215
Other payables	6012
Current assets	4345
Tangible fixed assets	3350
External debt	3016
Cash	2908
Equity	2872
Other receivables	2798
Trade receivables	1431
Trade payables	1254
Inventory	1231
Financial fixed assets	1174
Intangible fixed assets	945
Work in Progress	789
Corporate income Tax	759
Financial result	691
Provision	407
Deferred revenue	179

Table A.2: Classes Profit & Loss statement

Class	**Frequency**
Operational expenses	19580
Personnel expenses	10443
Revenue	7912
COS	7610
Financial result	4351
Depreciation	1811
CIT PL	823
Management fee	652
Result participations	618
Amortisation	343
Book result	52

Table A.3: Classes operational expenses

Class	**Frequency**
General expenses	9714
Sales expenses	1546
Car expenses	1502
Office expenses	1172
Housing expenses	1143
Facility expenses	487

Table A.4: Classes personnel expenses

Class	**Frequency**
Other personnel expenses	6436
Wages and salaries	1920
Social securities	856
Pension	435

Appendix B

Hyperparameter tuning results

B.1 Logistic Regression

This section shows all hyperparameter tuning experiments of the Logistic Regression model.

B.1.1 Profit & Loss statement

Figure B.1: Hyperparameter tuning experiment of the Logistic Regression model on the Profit & Loss dataset

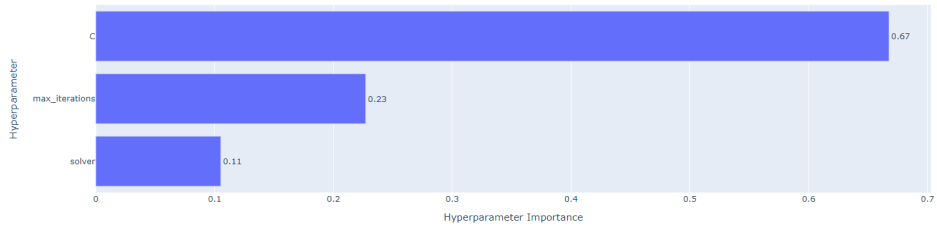

Figure B.2: Hyperparameter tuning importance of the Logistic Regression model on the Profit & Loss dataset

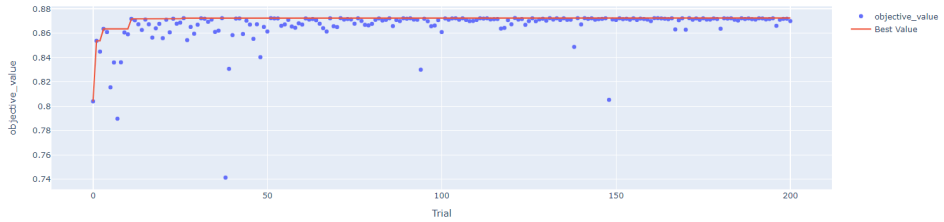

Figure B.3: Hyperparameter tuning results of the Logistic Regression model on the Profit & Loss dataset

B.1.2 Balance sheet

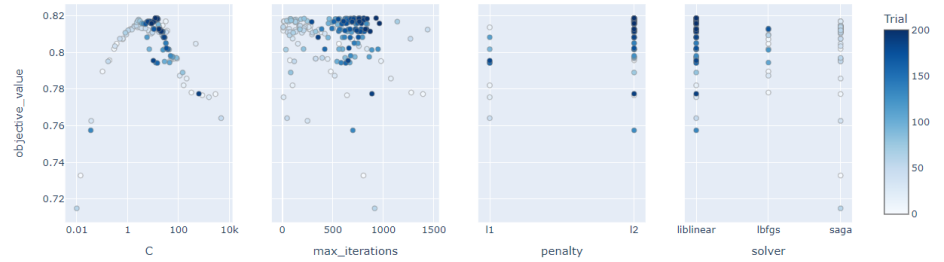

Figure B.4: Hyperparameter tuning experiment of the Logistic Regression model on the Balance dataset

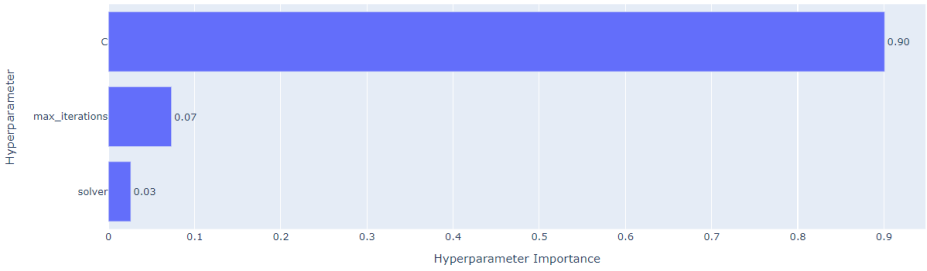

Figure B.5: Hyperparameter tuning importance of the Logistic Regression model on the Balance dataset

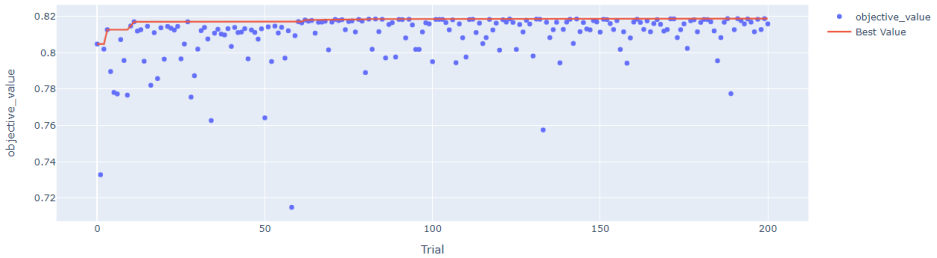

Figure B.6: Hyperparameter tuning results of the Logistic Regression model on the Balance dataset

B.1.3 Personnel expenses

Figure B.7: Hyperparameter tuning experiment of the Logistic Regression model on the Personnel Expenses dataset

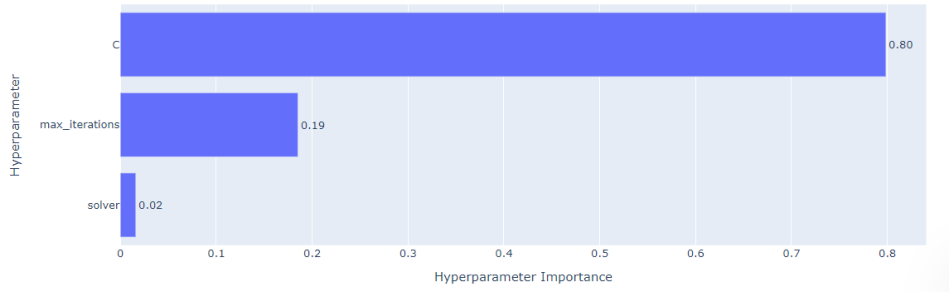

Figure B.8: Hyperparameter tuning importance of the Logistic Regression model on the Personnel Expenses dataset

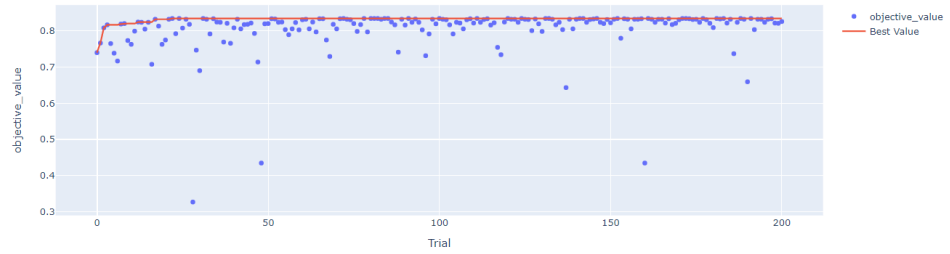

Figure B.9: Hyperparameter tuning results of the Logistic Regression model on the Personnel Expenses dataset

B.1.4 Operational expenses

Figure B.10: Hyperparameter tuning experiment of the Logistic Regression model on the Operational Expenses dataset

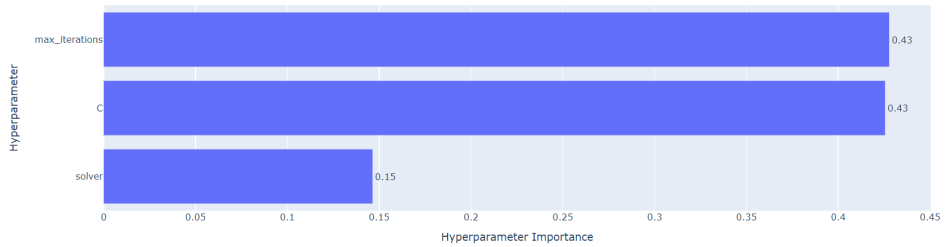

Figure B.11: Hyperparameter tuning importance of the Logistic Regression model on the Operational Expenses dataset

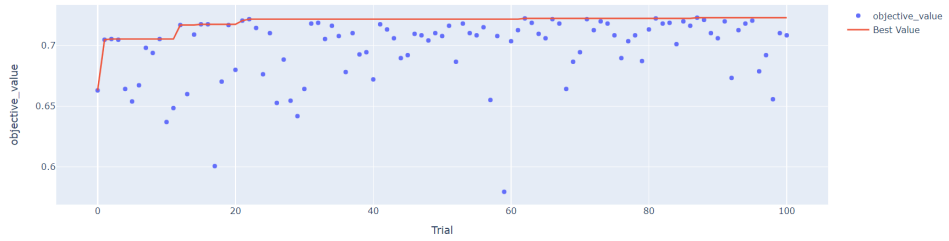

Figure B.12: Hyperparameter tuning results of the Logistic Regression model on the Operational Expenses dataset

B.2 Random Forest

This section shows all hyperparameter tuning experiments of the Random Forest model.

B.2.1 Profit & Loss statement

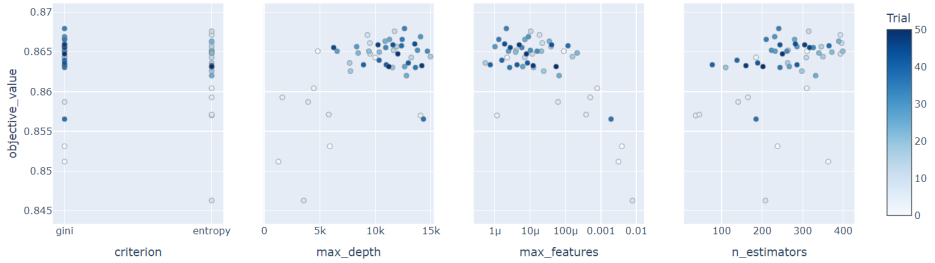

Figure B.13: Hyperparameter tuning experiment of the Random Forest model on the Profit & Loss dataset

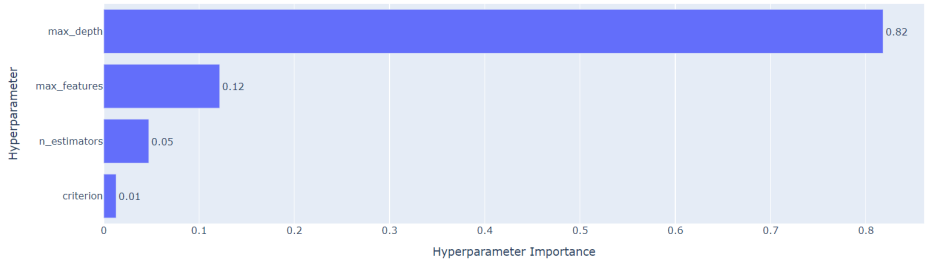

Figure B.14: Hyperparameter tuning importance of the Random Forest model on the Profit & Loss dataset

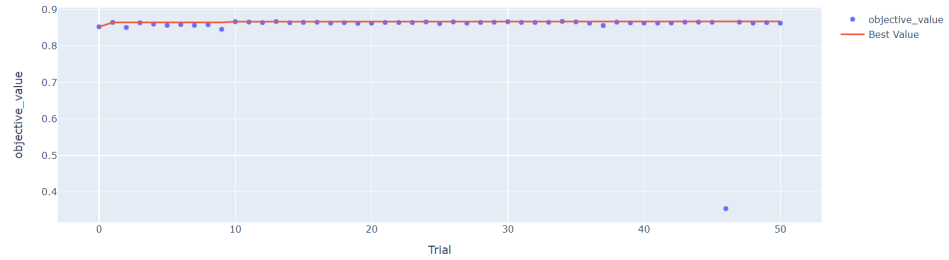

Figure B.15: Hyperparameter tuning results of the Random Forest model on the Profit & Loss dataset

B.2.2 Balance sheet

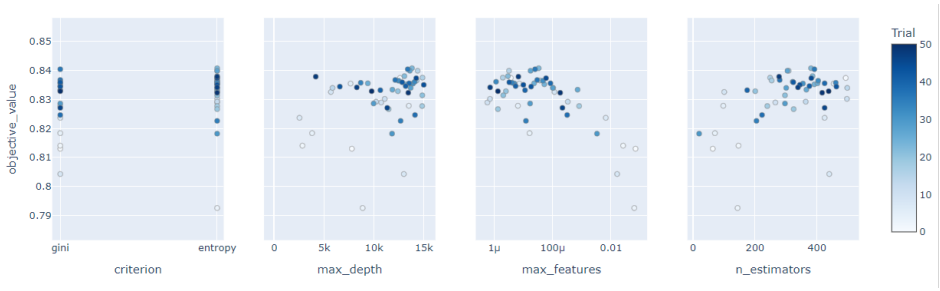

Figure B.16: Hyperparameter tuning experiment of the Random Forest model on the Balance dataset

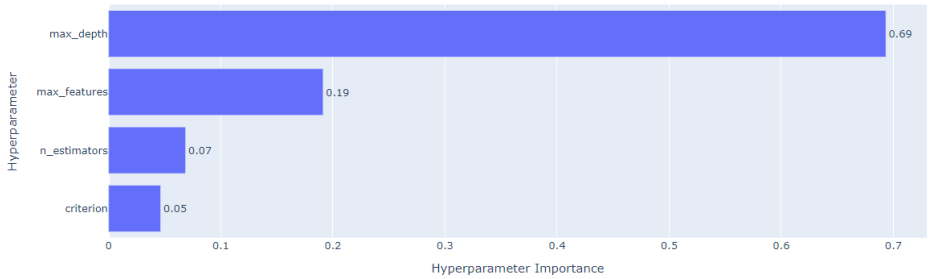

Figure B.17: Hyperparameter tuning importance of the Random Forest model on the Balance dataset

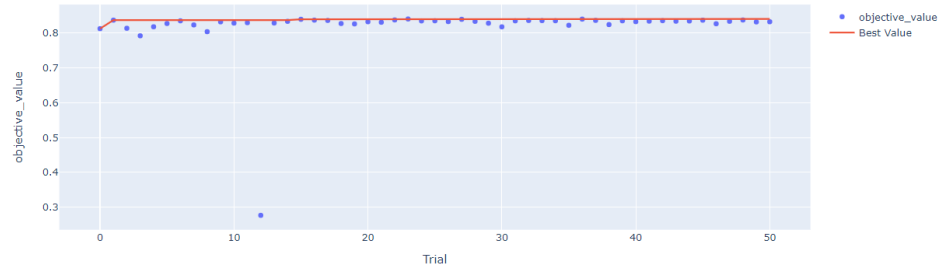

Figure B.18: Hyperparameter tuning results of the Random Forest model on the Balance dataset

B.2.3 Personnel expenses

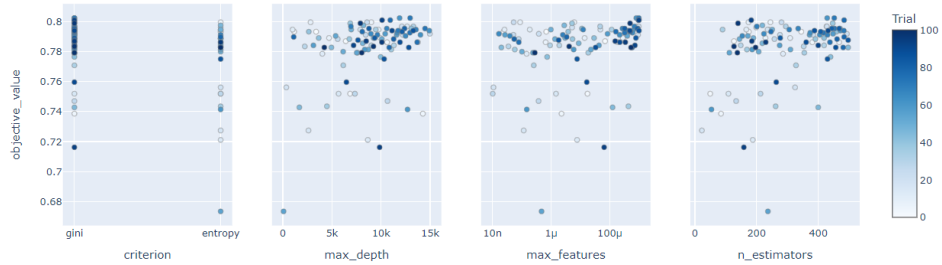

Figure B.19: Hyperparameter tuning experiment of the Random Forest model on the Personnel Expenses dataset

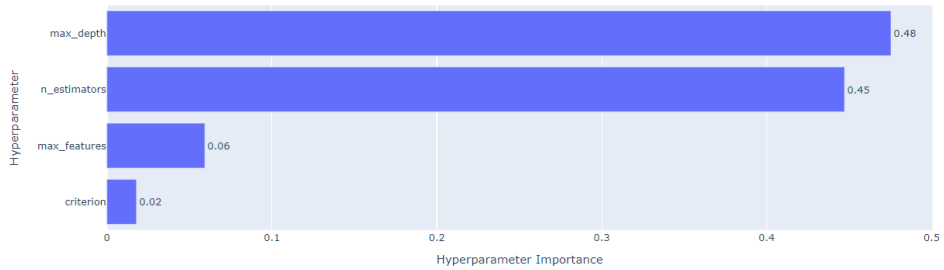

Figure B.20: Hyperparameter tuning importance of the Random Forest model on the Personnel Expenses dataset

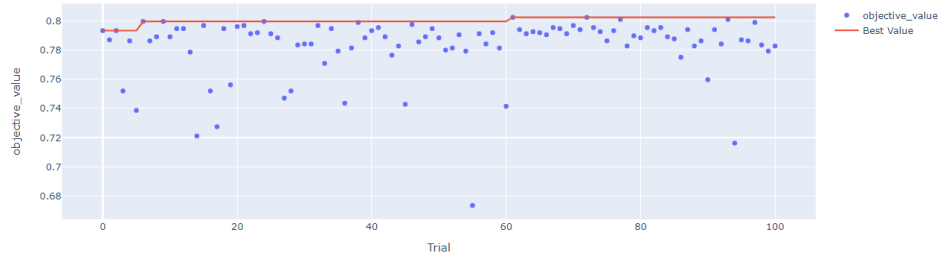

Figure B.21: Hyperparameter tuning results of the Random Forest model on the Personnel Expenses dataset

B.2.4 Operational expenses

Figure B.22: Hyperparameter tuning experiment of the Random Forest model on the Operational Expenses dataset

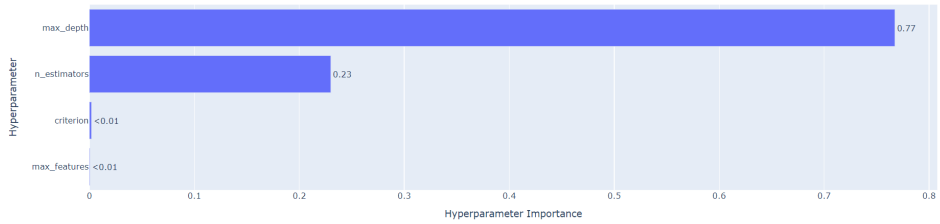

Figure B.23: Hyperparameter tuning importance of the Random Forest model on the Operational Expenses dataset

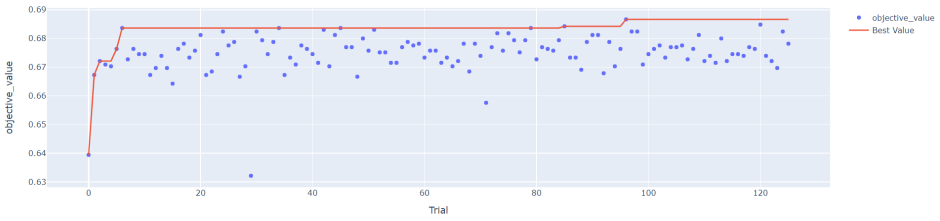

Figure B.24: Hyperparameter tuning results of the Random Forest model on the Operational Expenses dataset

B.3 Support Vector Machine

This section shows all hyperparameter tuning experiments of the Support Vector Machine model.

B.3.1 Profit & Loss statement

Figure B.25: Hyperparameter tuning experiment of the Support Vector Machine model on the Profit & Loss dataset

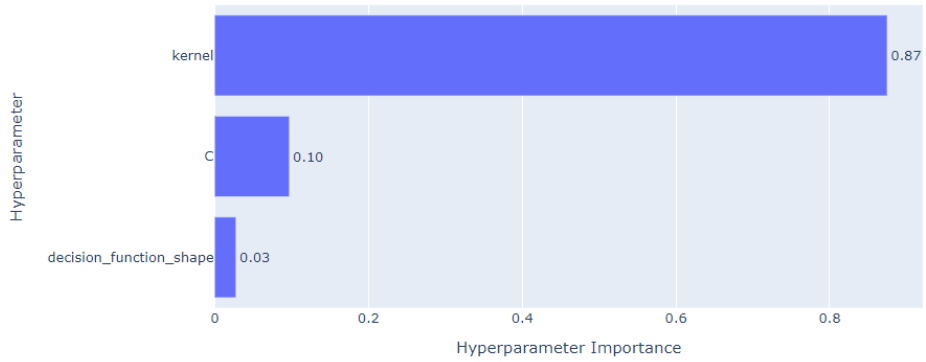

Figure B.26: Hyperparameter tuning importance of the Support Vector Machine model on the Profit & Loss dataset

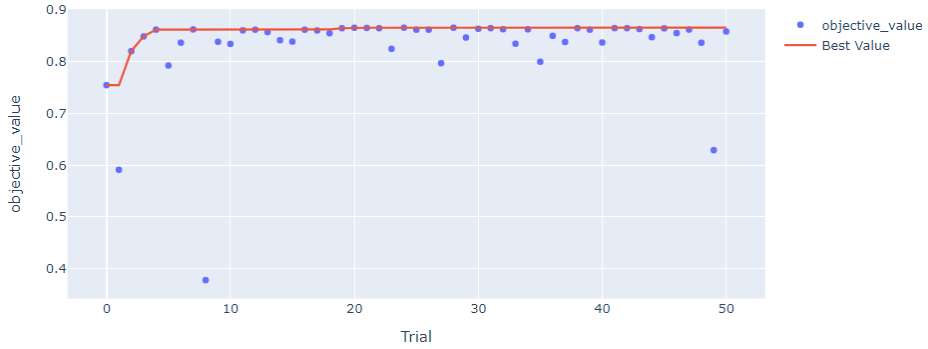

Figure B.27: Hyperparameter tuning results of the Support Vector Machine model on the Profit & Loss dataset

B.3.2 Balance sheet

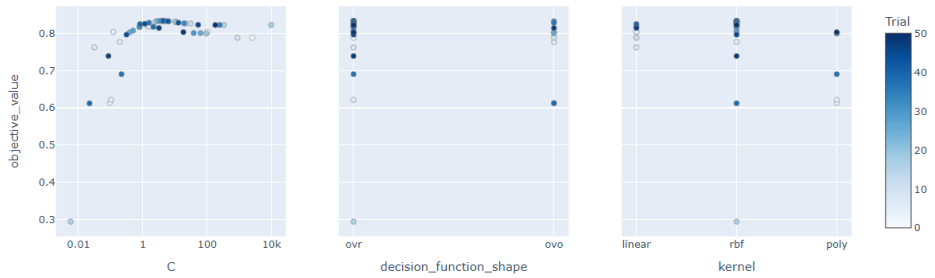

Figure B.28: Hyperparameter tuning experiment of the Support Vector Machine model on the Balance dataset

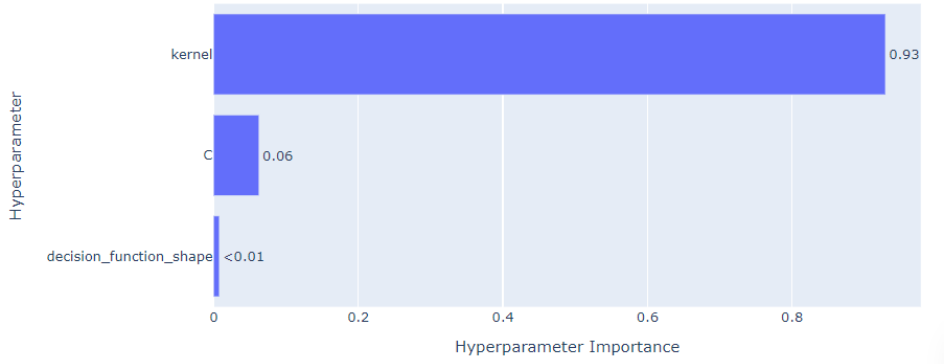

Figure B.29: Hyperparameter tuning importance of the Support Vector Machine model on the Balance dataset

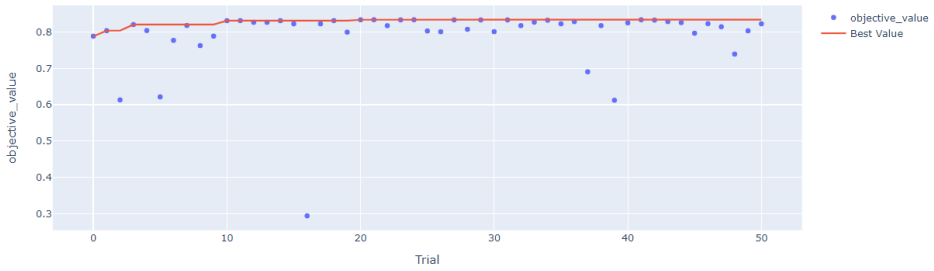

Figure B.30: Hyperparameter tuning results of the Support Vector Machine model on the Balance dataset

B.3.3 Personnel expenses

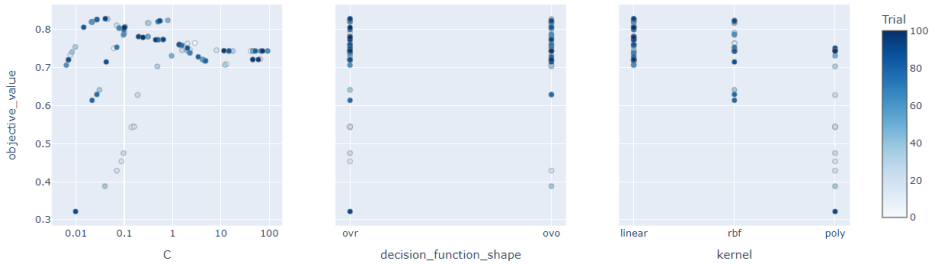

Figure B.31: Hyperparameter tuning experiment of the Support Vector Machine model on the Personnel Expenses dataset

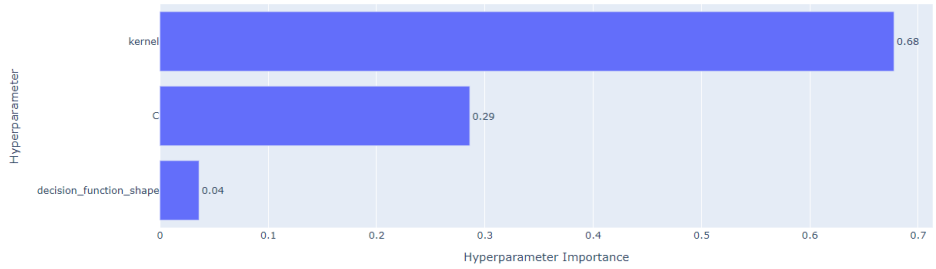

Figure B.32: Hyperparameter tuning importance of the Support Vector Machine model on the Personnel Expenses dataset

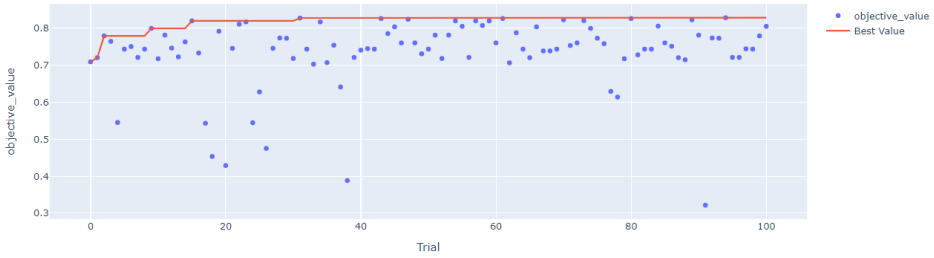

Figure B.33: Hyperparameter tuning results of the Support Vector Machine model on the Personnel Expenses dataset

B.3.4 Operational expenses

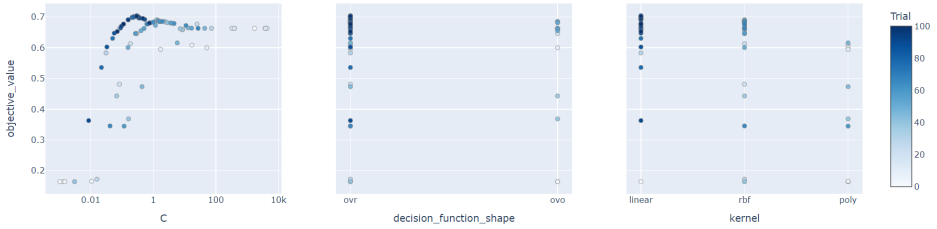

Figure B.34: Hyperparameter tuning experiment of the Support Vector Machine model on the Operational Expenses dataset

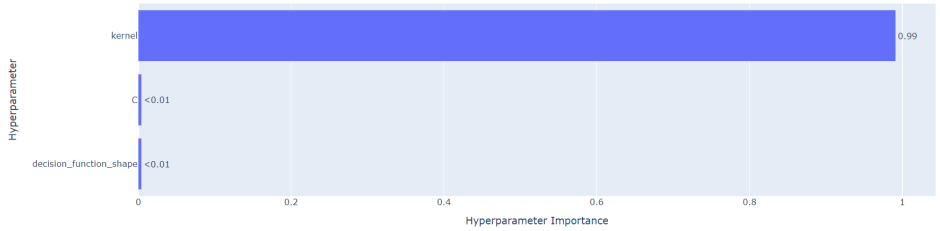

Figure B.35: Hyperparameter tuning importance of the Support Vector Machine model on the Operational Expenses dataset

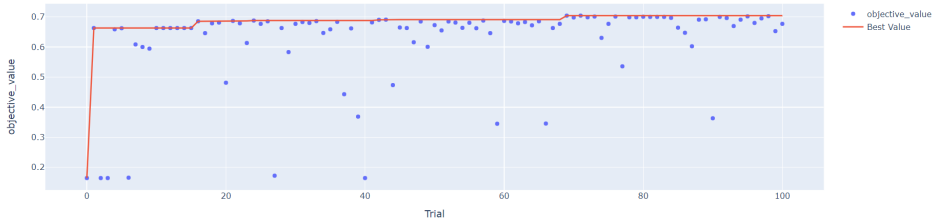

Figure B.36: Hyperparameter tuning results of the Support Vector Machine model on the Operational Expenses dataset

B.4 BERT

This section shows all hyperparameter tuning experiments of the BERT model.

B.4.1 Profit & Loss statement

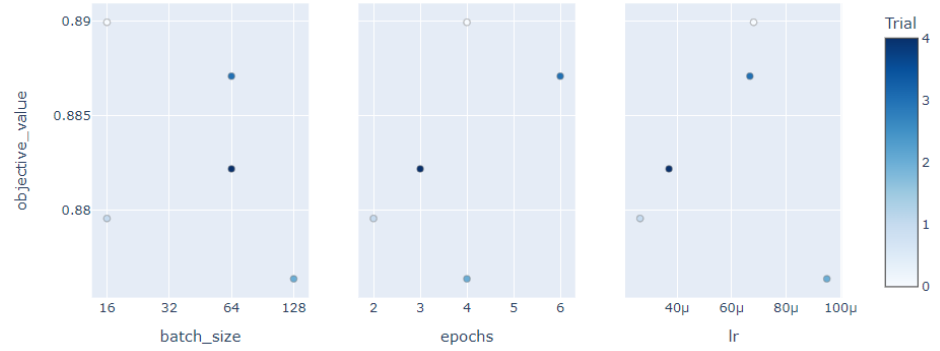

Figure B.37: Hyperparameter tuning experiment of the BERT model on the Profit & Loss dataset

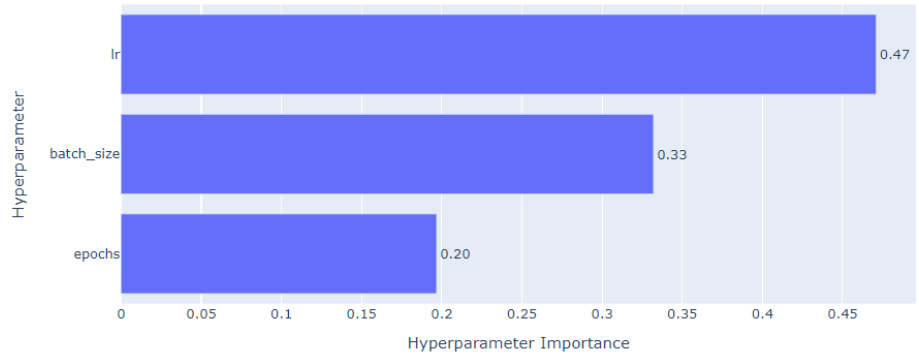

Figure B.38: Hyperparameter tuning importance of the BERT model on the Profit & Loss dataset

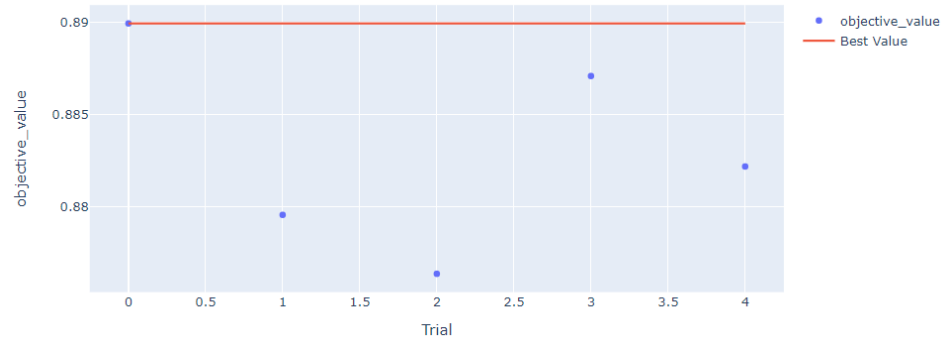

Figure B.39: Hyperparameter tuning results of the BERT model on the Profit & Loss dataset

B.4.2 Balance sheet

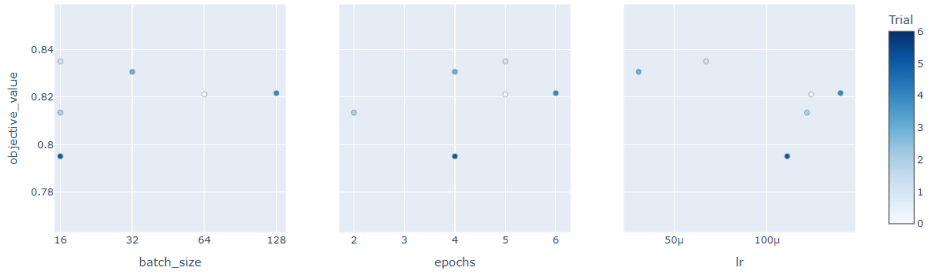

Figure B.40: Hyperparameter tuning experiment of the BERT model on the Balance dataset

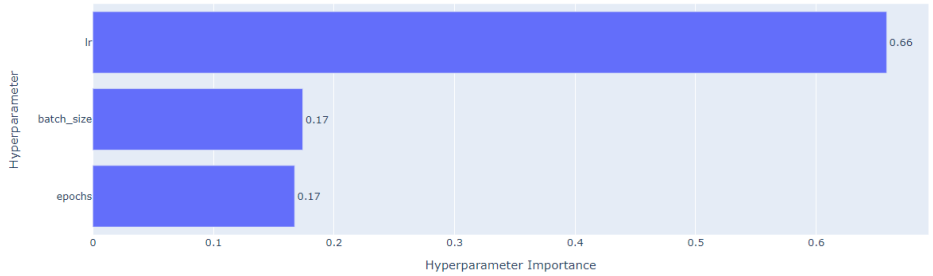

Figure B.41: Hyperparameter tuning importance of the BERT model on the Balance dataset

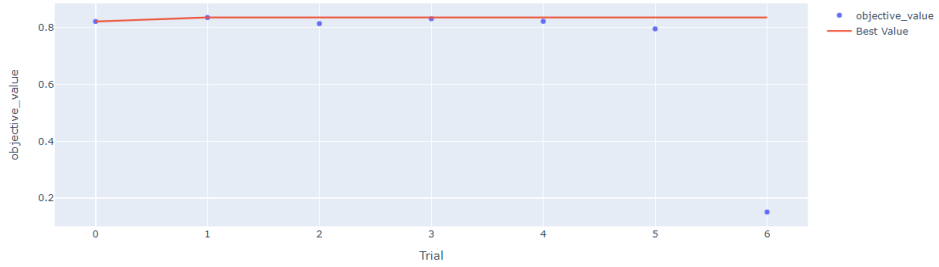

Figure B.42: Hyperparameter tuning results of the BERT model on the Balance dataset

B.4.3 Personnel expenses

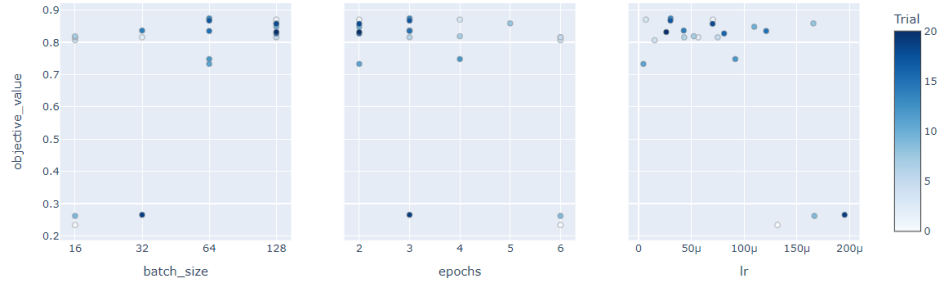

Figure B.43: Hyperparameter tuning experiment of the BERT model on the Personnel Expenses dataset

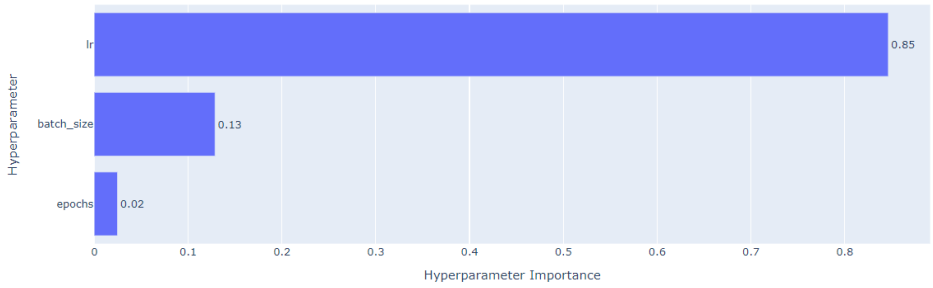

Figure B.44: Hyperparameter tuning importance of the BERT model on the Personnel Expenses dataset

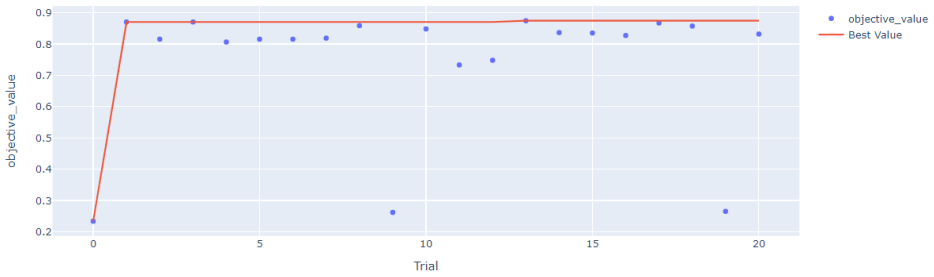

Figure B.45: Hyperparameter tuning results of the BERT model on the Personnel Expenses dataset

B.4.4 Operational expenses

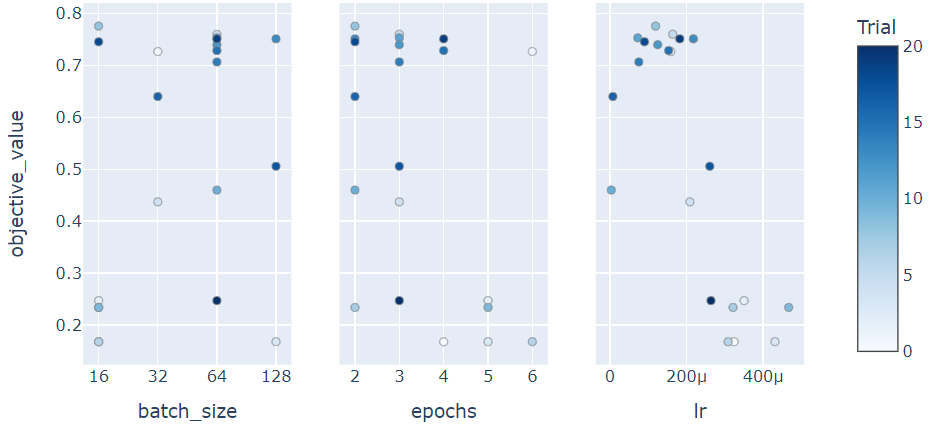

Figure B.46: Hyperparameter tuning experiment of the BERT model on the Operational Expenses dataset

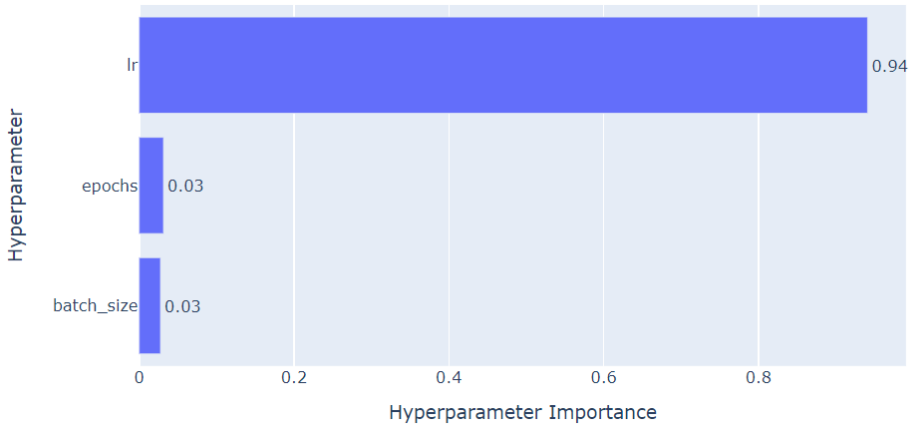

Figure B.47: Hyperparameter tuning importance of the BERT model on the Operational Expenses dataset

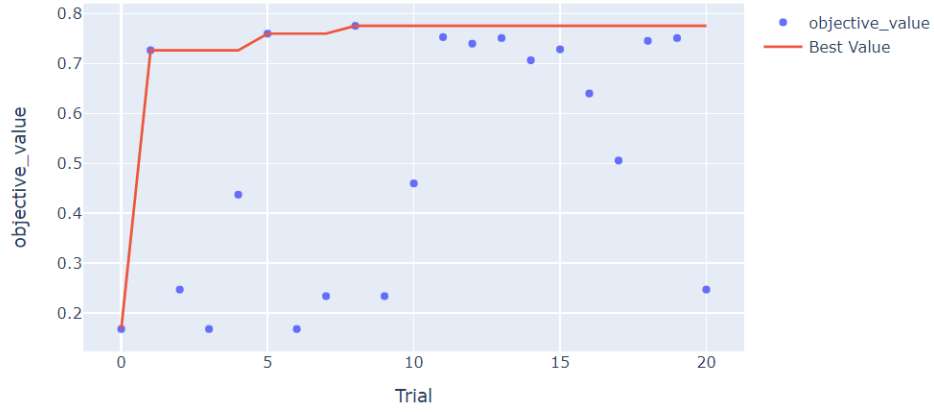

Figure B.48: Hyperparameter tuning results of the BERT model on the Operational Expenses dataset

B.5 BERTje

This section shows all hyperparameter tuning experiments of the BERTje model.

B.5.1 Profit & Loss statement

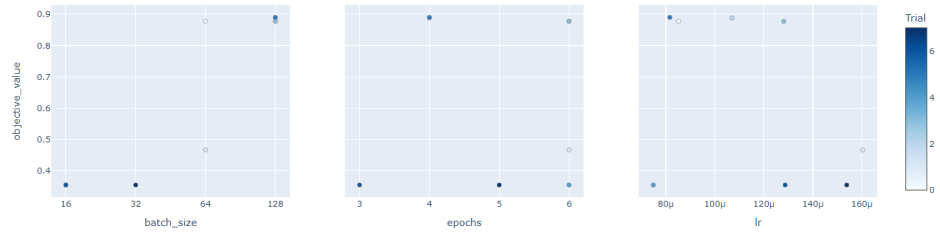

Figure B.49: Hyperparameter tuning experiment of the BERTje model on the Profit & Loss dataset

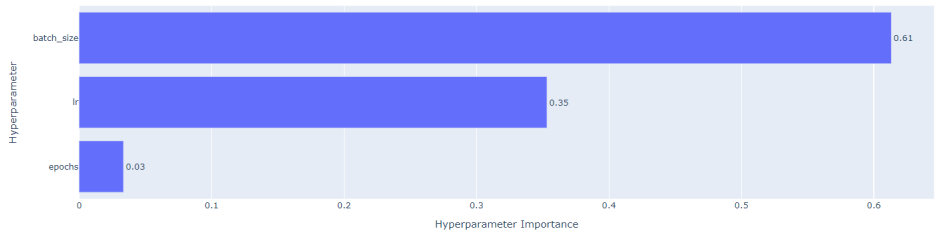

Figure B.50: Hyperparameter tuning importance of the BERTje model on the Profit & Loss dataset

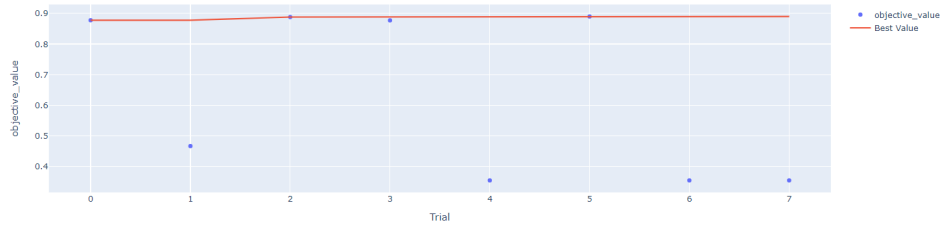

Figure B.51: Hyperparameter tuning results of the BERTje model on the Profit & Loss dataset

B.5.2 Balance sheet

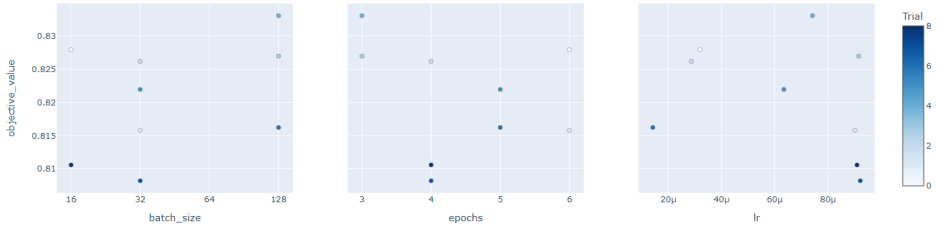

Figure B.52: Hyperparameter tuning experiment of the BERTje model on the Balance dataset

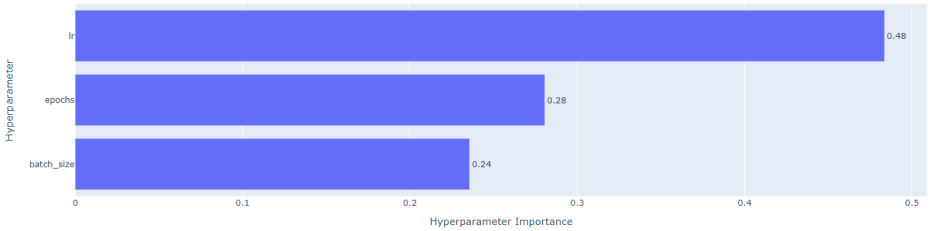

Figure B.53: Hyperparameter tuning importance of the BERTje model on the Balance dataset

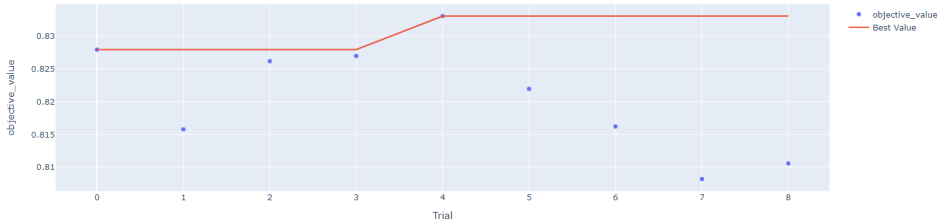

Figure B.54: Hyperparameter tuning results of the BERTje model on the Balance dataset

B.5.3 Personnel expenses

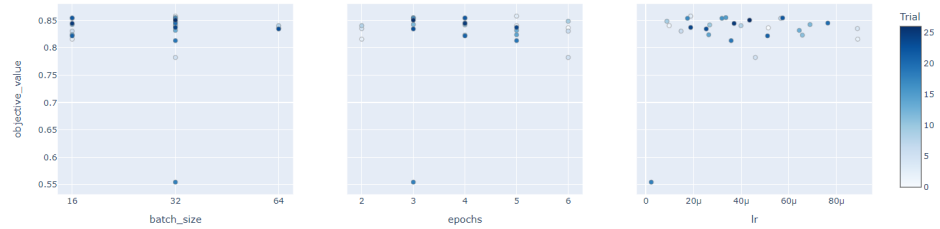

Figure B.55: Hyperparameter tuning experiment of the BERTje model on the Personnel Expenses dataset

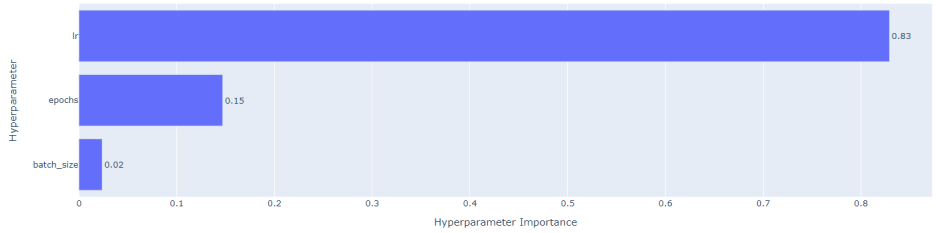

Figure B.56: Hyperparameter tuning importance of the BERTje model on the Personnel Expenses dataset

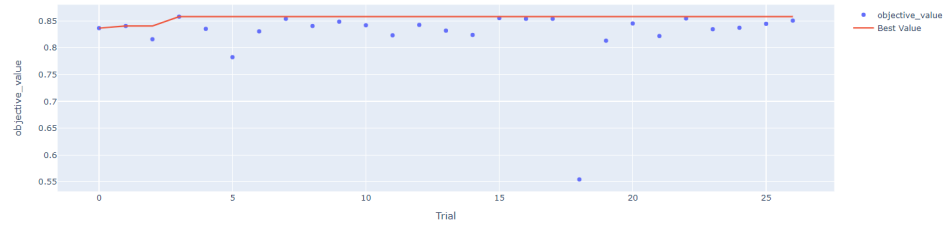

Figure B.57: Hyperparameter tuning results of the BERTje model on the Personnel Expenses dataset

B.5.4 Operational expenses

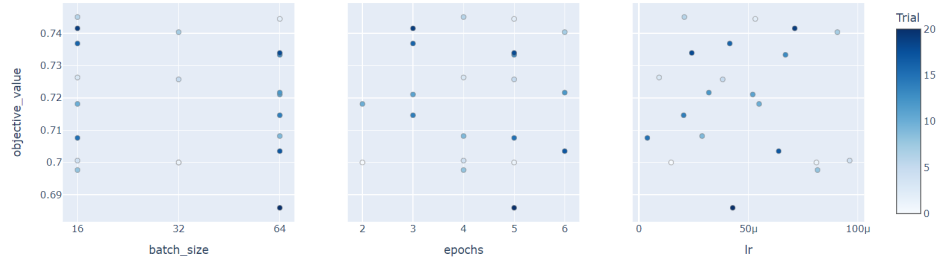

Figure B.58: Hyperparameter tuning experiment of the BERTje model on the Operational Expenses dataset

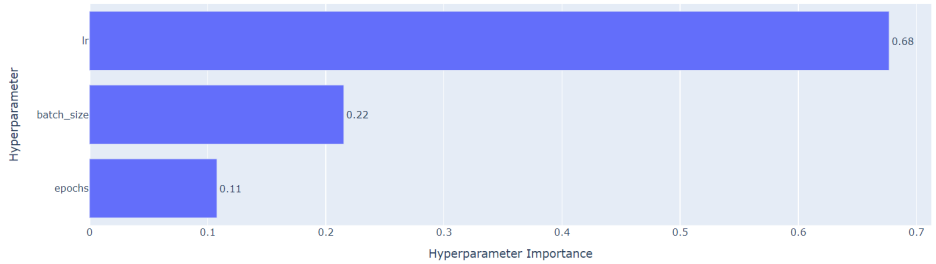

Figure B.59: Hyperparameter tuning importance of the BERTje model on the Operational Expenses dataset

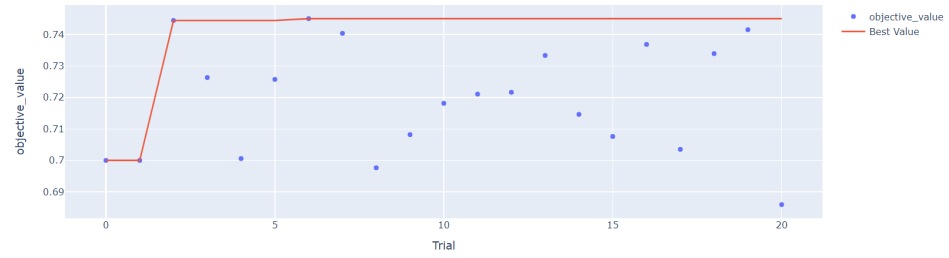

Figure B.60: Hyperparameter tuning results of the BERTje model on the Operational Expenses dataset