Erasmus School of economics

Master thesis Econometrics & Management Science

Business Analytics & Quantitative Marketing

April 17, 2024

Classifying the Payment Term of an Invoice using Supervised and Semi-Supervised Machine Learning Techniques

Author:

Sophie Prins (448848)

Supervisor:

M. van de Velden

Second assessor:

C. Cavicchia

Abstract

This research investigates methods for classifying payment terms in invoices using supervised and semi-supervised machine learning techniques. We analyze the effectiveness of different text representations and classification models, including K-Nearest Neighbors, Multinomial Logistic Regression, and Support Vector Machines. Our research demonstrates that the payment term of an invoice can be effectively classified into a finite number of classes using a feature set based on bigrams, which are dimensionality reduced using SVD, and employing a Multinomial Logistic Regression algorithm configured with the sag solver and a regularization parameter strength of 10. Our findings reveal that bigram-based features consistently outperform other representations, indicating the importance of capturing sequential dependencies in invoice text. Additionally, we explore the potential of semi-supervised learning to enhance model performance but find that the top-performing supervised model generally outperforms the semi-supervised learning models. Despite promising results, our study identifies several limitations and suggests future research directions, such as incorporating an infinite number of classes and exploring alternative algorithms. Overall, our research contributes to improving automatic invoice processing, with implications for businesses like Blue10.

Keywords: Invoice, Payment term, Textual Representations, Machine Learning, Super-vised Learning, Semi-Supervised Learning

Contents

1 Introduction

Accounting is the backbone of business. Accounting provides a business with financial insights from previous, current and future years. Blue10 is a Dutch company that helps organizations to automate administrative processes. These processes include digital processing of purchase and sales invoices, packing slips and receipts. The Blue10 software automates repetitive tasks and checks the quality and completeness of the administration. The goal of the company is to make accounting fully automated by the year 2032. In order to achieve this goal, information in invoices needs to be recognized by the Blue10 software. Data that can be retrieved from invoices are, among others, the vendor, invoice date, IBAN, chamber of commerce number, VAT number, total payment amount and payment term. Methods for recognizing and extracting the vendor, invoice date, IBAN, chamber of commerce number, VAT number, and total payment amount of an invoice are already available within the company's software. Currently, the methods to do this are developed mainly using rules and text patterns. As an illustration, extracting an IBAN can be easily accomplished using a text pattern, given that an IBAN follows a predefined structure. An IBAN usually starts with a two-letter country code, followed by two check digits and accommodating up to thirty alphanumeric characters, all with a predetermined length varying by country.

The payment term of an invoice refers to when its payment is due. This is usually relative to the invoice date or the date on which the services were delivered. In most cases, the payment term is stated directly in the invoice text or can be derived from other information in the invoice text. For example, the invoice text specifies the maximum amount of days within which the payment must be completed, or alternatively, the payment term corresponds to the number of days between the invoice date and the due date. Developing a method reliant on rules and text patterns to predict the payment term of an invoice poses challenges, given the substantial variations in invoice and payment term formats across different vendors. The absence of a standardized structure complicates the task. The accuracy of the predictions using a rule-based method depends on the complexity of the rules and the method's ability to capture patterns and factors that determine the payment term. Machine learning approaches trained on historical data are able to capture more complex patterns. The objective of this research is to find a method to classify the payment term of an invoice based on the invoice text. To achieve this we first need to consider how to process raw text, then given the text representation(s) how to classify the payment term best.

Most machine learning algorithms work with numerical data and cannot process raw text data. An invoice is a text document that consists of raw text data. Raw text data refers to text that has not yet been pre-processed or cleaned. There exist procedures that convert raw text data into a numerical representation. These procedures usually consist of pre-processing the text data, for example by removing stop-words like "the" and "is", and converting the text data into a numerical vector representation per text document. These vector representations can serve as input for a machine learning algorithm that aims to classify the payment term of an invoice.

Supervised classification in machine learning involves training a model with input objects and their associated desired output values. These desired output values are also known as classes or labels. The model learns from this training data to predict the output values for new input objects. In this case, classification models are trained using the numerical representations of invoices along with their corresponding labels. In our data set, invoices are labeled with their payment term. The classification models include K-Nearest Neighbors, Multinomial Logistic Regression and Support Vector Machines. Performance across models is evaluated by comparing their outputs with actual values and calculating statistics such as the F_1-score and Cohen's κ. The following two research questions capture the approach.

What procedure that converts raw text data into a numerical representation can best be used in a supervised classification model, K-Nearest Neighbors, Multinomial Logistic Regression or Support Vector Machines, to classify the payment term of an invoice according to the performance denoted by the F_1-score?

What algorithm out of K-Nearest Neighbors, Multinomial Logistic Regression and Support Vector Machines can best be used to classify the payment term of an invoice according to the performance denoted by the F_1-score?

As previously mentioned, supervised classification algorithms rely on labeled data. Acquiring labeled data for the problem at hand, classifying the payment term of an invoice, is expensive and time-consuming. However, unlabeled data are readily available. When limited labeled data and a lot of unlabeled data are available, semi-supervised learning algorithms can be appropriate for the classification problem at hand. Especially, if the unlabeled data contain valuable information for predicting classes. The information is valuable when it is not present in the labeled data and it contributes to determining the class of an observation. Furthermore, unlabeled data can improve the generalization performance of the model. Particularly, when the labeled data are not representative of the population. In semi-supervised learning, a model is trained on both labeled and unlabeled data. For that reason, the following research question is:

Can semi-supervised learning improve the performance reported by the F_1-score of payment term classification for invoices, compared to the performance of the supervised learning models?

Frey and Osborne (2017) examined how likely jobs are to be automated. They concluded that the accountancy profession will be highly automated in the future. Examining invoices manually is sensitive to errors and is time-consuming. Providing customers of Blue10 with the correct recognized data saves them time, money and resources.

The main relevance of this research can be divided into two fields, namely the relevance for Blue10 and the scientific relevance. First, the relevance for Blue10 is explained. Results of and methods developed within this research can be used to enrich the Blue10 software and therefore serve the Blue10 customer. The solution enables Blue10 to classify the payment term of invoices. Currently, when processing an invoice through Blue10, the payment term that is assigned in the accounting system is not based on the payment term specified in the invoice itself. Instead, it is determined by the vendor-specific settings configured by the users within the accounting system. The real payment term may, in fact, be different. For example, consider a scenario where the invoice explicitly indicates a payment term of thirty days. However, due to the predefined payment term set by the accountant for the specific vendor, which is fourteen days, the invoice will be recorded in the accounting system with a payment term of fourteen days.

The payment term, and therefore the moment of payment, could influence a company's liquidity position, financial planning and risk assessments. Furthermore, payment terms play a significant role in supplier relations. Consistent payment practices can build trust, support suppliers' cash flow, create a competitive advantage and enhance the overall reputation of a company. It is important for companies to carefully consider payment terms.

Second, the scientific significance is elaborated upon. Research investigating the application of semi-supervised machine learning algorithms for classifying the payment term of invoices using raw text data is limited. Previous studies have not extensively explored the development and evaluation of an optimal model designed and trained for the sole purpose of classifying the payment term of an invoice, regardless of whether the model is semi-supervised or supervised. Thus, there is a notable research gap in understanding the effectiveness and performance of such models in accurately classifying the payment term of an invoice.

The rest of this paper is structured as follows. The following section provides a literature review focusing on techniques for converting text data into numerical representations, as well as semi-supervised learning techniques and automation methods found in the literature concerning invoice processing. Section 3 describes the data used in this research. Hereafter, the methodology is described in Section 4. We present the results in Section 5. The last section gives a conclusion and discussion. Suggestions for future research either on a scientific level or company level are also provided in this section.

2 Literature Review

The tasks performed by an accountant are mainly repetitive tasks. Machine learning techniques can be used to reduce and automate repetitive tasks. A task that can be automated is the recognition of specific information present in an invoice, for example, the invoice date, IBAN or payment term. The objective of this study is to develop an automated method for identifying the payment term associated with an invoice. In this section, we first address the literature on textual representations, followed by an overview of semi-supervised learning methods. Subsequently, we investigate automation methods applied in invoice processing.

2.1 Textual Features

A corpus is a large collection of documents or text pages. The data that we consider for classifying the payment term of an invoice are raw text data, namely the text present in invoices. However, text data poses challenges regarding its automatic processing due to the ambiguity and unstructured nature of the data. The textual representation of an invoice must be converted to a numerical input vector to serve as input for a machine-learning algorithm, as machine-learning algorithms process numerical features to make predictions or classifications. A first step of converting textual data to numerical vectors is breaking down text data into individual tokens, which is called tokenization. These tokens are sequences of characters or words. After the text has been tokenized, the tokens are converted into numerical representations.

2.1.1 Bag of Words

A *Bag-of-Words* (hereafter BoW) representation can be used to represent text data as numerical input vectors. The representation involves a vocabulary of tokens in the corpus and a measure of the presence of a token. The vocabulary of tokens consists of (a selection of) unique tokens from the corpus. Once a vocabulary is chosen, a matrix where each entry measures token presence in a text document is created. The resulting matrix is called the *Document-Term matrix.* In its simplest form, the presence of a token is measured with a binary indicator, a token is either present or absent. More sophisticated measures use raw counts or differently computed measures.

The BoW representation is a simple and orderless representation of the documents (Radovanović & Ivanović, 2008). In this context, ”orderless” means that each document is treated as collection (bag) of words without preserving their specific order. Information about the structure of words is discarded. Instead, it focuses on the occurrence of individual words within the text. This has some limitations, such as the loss of contextual information and the inability to capture the relationships between words (Dharma, Gaol, Warnars, & Soewito, 2022).

A BoW representation can be created for a sequence of N tokens (*N-grams*). For example, a 1-gram, also known as a unigram, equals a single token extracted from a tokenized text. When N increases, the number of N-grams present in the corpus typically increases exponentially. For tasks that require understanding the text, such as topic modeling or document classification, a larger N might be more appropriate as it captures longer phrases and semantic context. In contrast, for tasks that require less understanding of the text, such as spell correction or named entity recognition, a smaller N might be more suitable.

Naturally, some tokens appear more often than others and have a high frequency regardless of the class. Because these tokens are common across multiple classes, they don't provide much information for distinguishing between the classes. These common tokens, despite not being distinctive in the classes, can still influence the prediction of a document's class label due to random variations in how they are distributed across different classes within the dataset. These tokens should be given less weight in a classification model. The *Term Frequency Inverse Document Frequency* (hereafter TF-IDF) is a measure that can reflect how relevant a token is to a document in the corpus. The resulting value reflects both the importance of the token within the document and its importance across the corpus. It can be used as a weighting measure which is applied to obtain more sophisticated BoW representations. TF-IDF is a result of research conducted by Luhn (1957) and Sparck Jones (1972), and equals the product of two measurements. The first part is the token frequency (TF), which represents the number of times a token appears in a document, a row in the Document-Term matrix. The second part measures the rarity of a token across the entire corpus (IDF), all rows in the Document-Term matrix. The value of IDF increases for tokens that occur infrequently in the corpus and decreases for tokens that occur frequently. A high TF-IDF value for a particular token suggests that the token is both frequently used within that specific document and is relatively rare across the corpus. This combination of high frequency within the document and relative rarity across the corpus indicates that the token is important and distinctive within that specific document, potentially carrying significant meaning or relevance to the document's content.

Dimensionality Reduction The BoW representations are often high dimensional sparse document representations. That is, each document is described by a large number of features, most of which are zero-valued. If the machine learning algorithms provided with these document representations cannot scale to such high dimensions, then the machine learning algorithms can experience decreasing performance. Furthermore, high dimensional sparse representations may hinder the application of the algorithms due to technical reasons. Specifically, as the dimensions increase, the computational and memory resources required to process and analyze the data also increase (Radovanović & Ivanović, 2008).

Dimensionality reduction is the process of reducing the dimensions while preserving the most important information. *Singular Value Decomposition* (hereafter SVD) is a matrix decomposition that can be used to reduce the dimensionality of a dataset. The SVD theorem states that any real-numbered matrix A can be decomposed as the product of three other matrices. The SVD of a matrix A can be found in Equation 1:

$$A = U\Sigma V^T \tag{1}$$

In our context, the matrix $A_{n\times m}$ is the Document-Term matrix. U and V are orthogonal matrices, with columns equal to the orthogonal eigenvectors of AA^T and A^TA, respectively. The matrix Σ is a diagonal matrix, where the diagonal elements are the square roots of the eigenvalues of AA^T. These diagonal elements, referred to as singular values, are usually arranged in descending order. The larger the singular value, the more variability in the data is captured by the corresponding singular vector. Selecting the top k singular values allows us to approximate the matrix A by multiplying the truncated matrices $U^*_{n\times k}$, $\Sigma^*_{k\times k}$ and $V^{*T}_{k\times m}$. For SVD reduction we choose matrix V^* consisting of columns of V corresponding to the k largest singular values and when the original Document-Term matrix A is multiplied with this matrix V^*, this yields a new matrix A^* in a lower dimensionality.

Radovanović and Ivanović (2008) state that applying dimensionality reduction on BoW data, such as the Document-Term matrix, results in columns that are "topics". These new topics obtained by the reduction are obtained from combinations of the original tokens. Dimensionality reduction transforms the initial high-dimensional Document-Term matrix into a lower-dimensional space, capturing latent topics within the data. Textual data can contain complex relationships and meanings that are not immediately apparent. Latent topics are not explicitly expressed in the text but are inferred based on the co-occurrence patterns of words within the data. These topics are derived from the relationships between tokens appearing together in various documents. In the Document-Term matrix, each column corresponds to a unique token present in the corpus vocabulary, while each row represents a document in the corpus. The three matrices obtained when applying SVD to a Document-Term matrix are: U, Σ, and V^T, where U contains the left singular vectors, capturing the relationships between documents and latent topics. Σ holds the singular values, representing the importance of each latent topic, and V^T contains the right singular vectors, which encode the relationships between tokens and latent topics.

Each column in the matrix V^T corresponds to a token in the corpus vocabulary, and each row represents a latent topic. The elements in each column cell indicate the strength of association between the token and each latent topic. Therefore, the columns in this matrix can be interpreted as token embeddings. Token embeddings represent individual tokens as real-valued vectors. Each column represents a token as a dense vector in a vector space, where the dimensions capture different latent topics. These token embeddings encode the relationships between tokens based on their co-occurrence patterns in the corpus.

In the resulting lower-dimensional Document-Term matrix, each row represents a document, while each column represents a latent topic. The elements in each row indicate the strength of association between the document and each latent topic captured by SVD. Therefore, the rows in this matrix can be interpreted as document embeddings. These embeddings encode the content of documents based on their relationships with latent topics extracted from the corpus to real-valued vectors.

Meyer (2000) provides a clear explanation of SVD along with an example illustrating its application to a Document-Term matrix. Meyer (2000) emphasizes that since the dimensions of a reduced matrix obtained through SVD are linear combinations of the input terms, there is a possibility of losing the clear distinction between documents that would be evident when considering the original tokens themselves. This issue becomes more prominent when a separation on a fine level is desired for a diverse set of documents, for example distinguishing documents about *corn* and *wheat* (Albright, 2004).

2.1.2 Word2Vec

Word2Vec refers to a collection of models that are also used to generate *word embeddings*. A word embedding represents a word as a numerical vectors in a predefined vector space, where words that are close to one another in meaning have a similar representation, meaning they are close to one another in the vector space. These word embeddings are commonly expressed as real-valued vectors spanning tens or hundreds of dimensions. In contrast to the high dimensions observed for the sparse Document-Term matrices, which can reach up to thousands or even millions of dimensions. A Word2Vec word embedding captures the meaning of a word and its relationship with other words within the corpus. In classification tasks like ours, classifying the payment term of an invoice, word embeddings have the advantage of retaining predicting power without the need for omitting or approximating variables. Compared to sparse vectors, although they are short, dense vectors are more meaningful.

The Word2Vec model structure was developed by Mikolov, Sutskever, Chen, Corrado, and Dean (2013) at Google. Word2Vec models are shallow, two-layer neural networks trained on a text corpus. The models construct a vector space where each distinct word in the corpus is assigned a corresponding vector in the space. Two different neural model architectures are part of the Word2Vec framework and can be used to learn word embeddings, namely *Continuous BoW* (hereafter CBoW) and *Continuous Skip-Gram* (hereafter Skip-Gram). The Skip-Gram model takes a target word as input and aims to predict the surrounding words within a specified window. The CBoW model takes a window of surrounding words as input and aims to predict the target word. In both architectures, a softmax activation function is utilized in the output layer. The softmax activation function is used to produce a probability distribution over the vocabulary. It assigns higher probabilities to words that are more likely to appear in the context words for a Skip-gram model or as the target word for a CBoW model.

For both Skip-gram and CBoW models, the output of the network is not of interest as the word embeddings are learned in the projection layer (hidden layer). The desired word embeddings are represented by the weights of the hidden layer. Mikolov (2013) stated that when provided with a substantial amount of training data, both implementations exhibit comparable performance in achieving their prediction objectives. However, when less data is available, Skip-Gram performs better and gives a good representation of rare words. CBoW, on the other hand, trains faster. Both architectures ignore the word order which accelerates the training process (Mikolov, Chen, Corrado, & Dean, 2013).

2.1.3 GLOVE

Pennington, Socher, and Manning (2014) introduced an embedding model called *Global Vector* (hereafter GLOVE). GLOVE incorporates word co-occurrences to obtain word vectors. It is a model that learns the relationship of words by investigating how often words appear close to each other in the given corpus. The objective of GLOVE is to learn word embeddings that minimize the difference between the dot product of the two word vectors and the logarithm of their co-occurrence probability. To achieve this, GloVe defines an objective function that equals the sum of the squared errors between the dot product of word embeddings and the logarithm of the co-occurrence counts. The objective function can be found in Equation 2.

$$J = \sum_{i,j=1}^{V} f(X_{i,j})(w_i^T \widetilde{w}_j + b_i + \widetilde{b}_j - log(X_{i,j}))^2 \tag{2}$$

In Equation 2, X denotes the word co-occurrence matrix, that represents the frequency of word co-occurrences within a defined context window. $X_{i,j}$ represents the number of times word W_j occurs in the context of word W_i. The vector w_i represents the word embedding in the word space for word W_i and $\widetilde{w}_j$ is the word embedding in the context space for word W_j. Additionally, b_i denotes the bias term in the word space associated with word W_i, and $\widetilde{b}_j$ stands for the bias term in the context space associated with word W_j.

The function $f(\cdot)$ is a weighting function typically used to reduce the influence of highly frequent word pairs. The term $log(X_{i,j})$ ensures that the model captures the logarithmic relationship between the word co-occurrences and the inner products of the vectors.

During training the objective function is minimized using an iterative optimization algorithm to learn the word vectors, as well as the bias terms. By construction, both GLOVE and Word2Vec models are unable to produce embeddings for out-of-vocabulary words. These out-of-vocabulary words are words that the models have not encountered during training and, therefore, do not have any numerical representation.

2.1.4 FastText

The *FastText* algorithm, developed by Facebook, is designed to overcome this limitation of not being able to produce embeddings for out-of-vocabulary words. Unlike Word2Vec and GLOVE, which consider words as indivisible, FastText breaks down each word into all possible character N-grams before learning their representations.

A character N-gram is a sequence of N characters. For example, the word "book" can be broken down into the character 2-grams: "bo", "oo", and "ok". The word itself is also included in the set of its N-grams, to learn a representation for each word. The architecture of this algorithm is similar to that of Word2Vec's CBoW or Skip-gram. FastText predicts the probability of a target word given its context, or vice versa, taking into account both word and sub-word information. To obtain a word-embedding out of the set of N-gram embeddings, the vectors are summed.

Fasttext represents words as a set of character N-grams in order to produce vector representations for both words that are and are not encountered during training. By utilizing subword-level embeddings, the algorithm can provide representations that capture the meaning and context of words not present in the corpus (Bojanowski, Grave, Joulin, & Mikolov, 2017). For example, we consider the words "playing" and "player". These words have the same root, "play," but have different grammatical forms. Even if "playing" is encountered in the corpus while "player" is not, there would still be a meaningful embedding for "player" due to the presence of "playing". Breaking down these words into character N-grams allows the embeddings to incorporate the common root "play," enabling them to establish a connection between "playing" and "player" despite their differing grammatical roles.

2.1.5 Document Representations

Word2Vec, GLOVE and Fasttext generate vectors for individual words. To derive a document embedding—a numerical representation capturing the semantic and contextual information of an entire document—from these word vectors, several methods have been suggested.

Schmidt (2019), for example, described a method that uses the TF-IDF measure in a weighting technique to improve upon the simple weighted sum of word vectors to obtain a dense vector for a document.

Le and Mikolov (2014) extended the Word2Vec model to also embed documents, this is referred to as *Doc2Vec.* Doc2Vec is a neural network-based model for generating document embeddings. The resulting vector is trained to capture the overall semantics of the document. Similar to Word2Vec, two different neural model architectures are part of the Doc2Vec framework, specifically the *distributed bag-of-words* model (hereafter DBOW) and the *distributed memory* model (hereafter DM). DBOW is a simpler model and can be compared to Word2Vec's Skip-gram implementation. In this approach, the focus lies on the document as a whole, the word order and the context words are disregarded. The document vectors are used to predict randomly sampled words from the document. DM works in a comparable way to CBoW, as DM aims to predict a target word given its context. However, in the DM framework, the document is seen as an additional input. The document vector is concatenated with the word vectors of the context words. This combined representation is then used to predict the target word.

Compared to BoW representations, Le and Mikolov (2014) found significant improvement in performance when using the Doc2Vec algorithm in information retrieval and sentiment analysis tasks. This suggests that Doc2Vec can be useful to capture the semantics of the input text. According to Le and Mikolov (2014), a combination of DM and DBOW is recommended due to its consistency across different tasks. However, DM performed better in capturing document semantics and context. Lau and Baldwin (2016) had contradictory results. They evaluated

Doc2Vec in two task settings. The first task is duplicate question detection in a web forum and the second task is predicting the similarity of a pair of sentences. Lau and Baldwin (2016) concluded that for both tasks, based on performance, DBOW is superior to DM.

2.1.6 Cross-Lingual Feature Representation

Words in a corpus may be from different languages. Thus, a feature generation method that incorporates a multi-lingual solution can be beneficial. Feature generation is the process of creating new input variables (*features*) from existing data. The goal is to create variables that better capture underlying patterns in the data. It can involve transforming existing variables, gathering new variables, or even creating new variables based on domain knowledge.

Van der Goot, Ljubešić, Matroos, Nissim, and Plank (2018) developed a feature generation method, called *bleaching text*. In this method, tokens in a text are transformed into abstract features using the token frequency, punctuation usage, vowel structure, shape, casing and length. The goal of the paper was to test whether abstract features perform well and are not restricted by the limitations of any single language but instead function effectively across a variety of languages. In order to do so, Van der Goot et al. (2018) designed experiments for predicting the gender of an author. In these experiments the abstract features are compared with features based on words and N-grams present in a given text, for both in- and cross-language. The experiments provide evidence that the abstract features enable models to function well when dealing with different languages. The abstract features ensure that a model is not restricted to understanding only one language. Instead, it has the ability to understand the shared underlying structures and meanings across different languages. In a uni-lingual case, features based on words and N-grams present in a given text perform the best. In a multi-lingual setting, this approach decreases in accuracy. The bleaching text model performs well in a multi-lingual setting and can identify certain hidden user characteristics by analyzing patterns in the data.

Van der Goot et al. (2018) compared the bleaching text approach with a model that uses multilingual embeddings developed by Plank (2017). Plank (2017) used the bilingual word embedding approach of Smith, Turban, Hamblin, and Hammerla (2017) and extended it to a multi-lingual setting, where words from multiple languages are mapped into a shared vector space. Van der Goot et al. (2018) showed that their solution is comparable to the multi-lingual word embeddings by Plank (2017).

2.1.7 BERT

The previously discussed embedding techniques are static, meaning that for each word or sequence of words only one vector exists. Alternatives that incorporate the fact that words can have multiple meanings are called bidirectional transformer encoders and lie at the basis of the *BERT* model developed by Google (Devlin, Chang, Lee, & Toutanova, 2018) or its modified document variant *DocBERT* (Adhikari, Ram, Tang, & Lin, 2019).

BERT, short for Bidirectional Encoder Representations from Transformers, relies on the Transformer architecture, a neural network architecture with multiple layers introduced by Vaswani et al. (2017). The Transformer consists of an encoder and a decoder. For BERT only the encoder is of importance. The encoder processes the input, such as a sentence, and converts it into a sequence of hidden representations. Each position in the input sequence is processed independently, allowing the model to capture contextual information. The Transformer utilizes a self-attention mechanism to evaluate the importance of words in a sentence during processing.

As opposed to language models that process text from right-to-left, or vice versa, BERT uses a bidirectional approach, which means that it processes the entire sequence of words at once. BERT is trained on two training objectives simultaneously, namely masked language modelling and next sentence prediction. In masked language modelling, words in the given text are randomly masked. Then, based on the context, the model learns to predict the original value of the masked word. In the next sentence prediction training process, the model receives pairs of sentences and learns to predict whether the second sentence follows the first sentence in the given text. BERT is typically trained on large amounts of unlabeled text to learn contextualized word embeddings. Hereafter, the trained BERT model can be adapted to perform well on a specific task like named entity recognition or text classification.

In order to obtain a document embedding using BERT, a document is first tokenized using BERT's tokenizer. Special tokens, denoted as [CLS] (for classification) and [SEP] (for separation), are then added to mark the beginning and end of the document, respectively. The tokens serve as an input for the BERT model. Each transformer layer processes the input sequences and produces hidden representations for each token. The representation of the start [CLS] token from the final transformer layer is trained to capture the overall meaning and context of the input sequence.

2.2 Semi-Supervised Learning

Two main practices have been discussed in the machine learning literature, specifically, supervised and unsupervised learning. In supervised learning, labeled data are present. The supervised learning methods attempt to generate a function with the labeled data, that can successfully determine the label of unseen data. For unsupervised learning, the label is unknown. Unsupervised learning methods map the data based on underlying patterns. In the case of a classification problem with scarce labelled data and lots of unlabelled data, creating a reliable supervised classifier can be challenging. In this case *semi-supervised learning* might be the solution.

Semi-supervised learning aims to combine both supervised learning and unsupervised learning. Labeling data can be time consuming, expensive or not even feasible due to the amount of data that are needed and/or available to produce an accurate classification algorithm. Semi-supervised learning takes advantage of all available data and attempts to construct a learner that outperforms learners that only make use of the labeled data. This is possible if the unlabelled data contain patterns or useful information for the predicting the label of an observation that is not available within the labelled data.

Van Engelen and Hoos (2020) present an overview of semi-supervised learning methods. The focus of the overview mainly lies on semi-supervised classification. In the paper more recent advances as well as earlier work are discussed. In our research, we make use of the wrapper methods as described by Van Engelen and Hoos (2020). In particular, Self-training and Co-training are implemented. Wrapper Methods are intuitive, straightforward and easy to implement. Furthermore, wrapper methods can be applied to any classification algorithm that can be trained on labeled data. This allows for direct comparison of performance with the supervised counterpart since the same classification algorithm can be used for both semi-supervised and supervised classification.

2.2.1 Wrapper Methods

Wrapper methods use one or more supervised classification algorithms in their process. This process involves two iterative steps: training and pseudo-labelling. First, labeled data are used to train the supervised classifiers. These classifiers are then used to predict the labels of the unlabeled data. The instances with the highest confidence scores are then labeled and added to the training set, this is called pseudo-labelling. The process is repeated until a satisfactory level of performance or a certain condition is achieved.

Self-training is the most simple approach among the wrapper methods, it only makes use of one supervised classifier (Triguero et al., 2017). Self-training was first used by Yarowsky (1995) in order to predict the meaning of a word based on its context. Design decisions for this method include the stopping criteria, the re-use of pseudo-labelled data and the selection of observations to be pseudo-labeled. The probabilistic predictions play an important role in the selection of observations to be pseudo-labeled. If the supervised classifier produces well-calibrated predictions, the approach is iterative and similar to the expectation-maximization algorithm (Dempster, Laird, & Rubin, 1977). Well-calibrated predictions refer to the situation where the predicted probabilities produced by a machine learning model match the true probabilities of the events being predicted. In other words, if the model predicts a probability of 0.8 for a certain event, then that event should occur approximately 80% of the time. When well-calibrated predictions are not available, for example when tree based algorithms are used, adaptions must be made in order to be able to use Self-training (Provost & Domingos, 2003).

Co-training is another approach among the wrapper methods in which two or more classifiers are trained that iteratively refine each other's predictions on the unlabeled data. The classifiers are iteratively trained. During each iteration, the most confident pseudo-labeled data generated by one classifier are added to the labeled dataset of the other classifier. The class predictions generated by the classifiers should demonstrate minimal correlation for a successful Co-training implementation. If the predictions are strongly correlated, then the potential that a classifier provides the other classifier with useful information is limited (Wang & Zhou, 2010).

Multi-view Co-training addresses this problem by exploiting the idea of training two classifier on two distinct feature sets. After each training step, the most confident predictions produced with the classifiers trained on each feature set are added to the labelled data for the classifier with the other feature set. This algorithm was first proposed by Blum and Mitchell (1998) and is applied in various fields, with particular emphasis on the field of computer science that focuses on enabling computers to process human language, known as *Natural Language Processing* (hereafter NLP) (Van Engelen & Hoos, 2020). Blum and Mitchell (1998) applied the multi-view Co-training algorithm to classify university web pages. They used the web page text and the text in the links to the web page from external sources to produce two distinct feature sets.

Co-training can be successfully implemented even when there is no apparent split in the feature set. In single-view Co-training, each classifier is trained on a different subset of features extracted from the same feature set, compared to multi-view Co-training where each classifier is trained on a different set of features that provides a different perspective of the data. Other single-view Co-training approaches focus on different ways to introduce diversity, such as using different classification algorithms or using different hyperparameters.

2.3 Machine Learning Techniques used in Invoice Processing

During invoice processing, machine learning algorithms have been applied in order to automate repetitive tasks. One such task involves the extraction of structured information from invoices.

Early approaches make use of document formats and positions to extract fields. The template of an invoice is either matched with previously processed templates (Schuster et al., 2013) or grouped based on similar properties (Hamza, Belaïd, Belaïd, & Chaudhuri, 2008; Esser, Schuster, Muthmann, & Schill, 2014). The desired information is then extracted using the matched observed field formats or rules. This extracted information is reliable when the documents are matched with the correct class or cluster, and thus the fields are extracted with the correct observed field formats and rules, but become unreliable when they are applied to unseen structures. Further, the number of groups and classes grows over time due to new vendors or appended formats and, therefore, this process is challenging to maintain. Format based methods, however, have the advantage of recognizing multiple fields all at once.

In order to achieve the advantage of detecting multiple invoice parts but being able to produce a method that generalises to unseen formats and that eliminates the idea of invoice classes, Aslan, Karakaya, Unver, and Akgül (2016) proposed a method for parsing invoices. The goal of this method is to identify and extract different parts of an invoice, such as line items, totals, and other relevant information. The method consists of two phases. The first phase employs various methods such as Support Vector Machines to generate candidate regions for the different type of invoice parts. In the next phase the detected candidate regions are utilized to determine the final positions. This last phase makes use of an optimization framework that allows for considering the interdependencies between the invoice parts and their relationship to the invoice as a whole.

Palm, Winther, and Laws (2017) developed a system called *CloudScan* that also has the capability to generalize to unseen invoice layouts. CloudScan utilizes a bidirectional long short-term memory network, a Recurrent Neural Network that effectively captures dependencies in sequential data from both past and future contexts. This model is trained on features derived from N-grams, along with labels provided by end-users. CloudScan focuses on extracting 8 fields typically found on invoices. The system is benchmarked against a Logistic Regression model. The models produced for unseen invoice layouts a F_1-score of 84% and 79%, respectively.

Named Enitity Recognition (hereafter NER) is a NLP task that involves locating and classifying specific named entities in a text. The goal of CloudScan, extracting key information, is very similar to performing NER. For that reason BERT for NER can be employed as well. Zhao, Niu, Wu, and Wang (2019) stated that applying BERT instead of the bi-directional long short-term memory network applied in *CloudScan* can achieve better results due to its superior performance. The latter model struggles to capture the relationship between distant words. Both models encounter difficulties in handling the variability of aligned text resulting from different invoice layouts. Zhao et al. (2019) proposed the Convolutional Universal Text Information Extractor (*CUTIE*), with the aim of incorporating spatial information into the key information extraction process to tackle the problem. CUTIE utilizes a Convolutional Neural Network that includes a word embedding layer to process text arranged in a grid structure. The grid containing invoice text is constructed using a grid positional mapping that takes the spatial relationship of the text into account. The word embedding layer effectively captures semantic information from the gridded text, allowing CUTIE to leverage both semantic and spatial information.

Other methods designed to extract key information incorporate characteristics of the specific fields. The *OCRMiner* developed by Ha and Horák (2022) incorporates, besides NER and other logical rules, key phrases. An example key phrase in the process of extracting the field *invoice date* is "invoice date:". This key phrase can be seen as a signal that the desired information can be found in the neighborhood content. This is combined with the use of the datatype of the desired field to extract the correct value. The choice of key words can, nevertheless, vary a lot. Liu, Zhang, and Wan (2016) used bags of potential features, that are constructed to capture properties for the 8 fields of interest to avoid the use of keywords. These are then weighted using Logistic Regression, Naïve Bayes and Support Vector Machine in order to classify every word group. Support Vector Machine produced the best result. Majumder et al. (2020) did not use key words either. They generated extraction candidates with the use of the data type of the target field. Then, the neighboring words of these candidates are used in a neural network system to score the candidates. The above mentioned models are, however, not models specifically designed for one of the target fields and are thus applicable for multiple entities present on invoices.

The previously mentioned research focuses on extracting multiple fields using either a single method or a method designed to generalize across multiple fields. However, research targeting the extraction of specific fields, such as payment terms, remain scarce. Topics that are closely related to payment term recognition, are the prediction of the moment of payment for an invoice and late payment predictions. Appel et al. (2019) compared the results of a Logistic Regression, Naïve Bayes, K-Nearest Neighbors, Random Forest and Gradient Boosted Decision Trees in a binary supervised classification problem and developed a predictive invoices label system based on an ensemble approach using Random Forest and Gradient Boosting. They attempted to predict whether an invoice will be paid *on time* or *late* and used features including *total invoices* *late*, *average days late* and *standard deviation invoices late*, in which they considered an invoice late when the payment occurred 5 days or more after the due date. The mentioned features as well as the classification problem itself, are dependent on the due date and thus the payment term. They achieved significant results with an accuracy up to 77%.

In the field of invoice processing, research has focused on various topics such as anomaly detection, invoice classification, invoice recognition, and predicting accounting journal entries.

Anomaly detection, which involves identifying abnormal behaviors like sudden invoice arrivals or invoices received at unusual times, presents challenges for manual detection. Tang et al. (2020) employed a machine learning-based technique to develop an anomaly detection method, achieving an accuracy exceeding 98%.

Invoice classification is another area of interest. Tarawneh, Hassanat, Chetverikov, Lendak, and Verma (2019) compared algorithms like K-Nearest Neighbors, Naïve Bayes, and Random Forest in categorizing invoices into types like *handwritten*, *machine-printed* and *receipts*, achieving an overall accuracy of 98.4% using K-nearest neighbors. Another study by Jadli and Hain (2020) compared combinations of pre-trained neural network's, dimension reduction techniques and classifiers, in a comparable task, and achieved a classification rate of 96.1% by using a full feature set and combining a Logistic Regression with a model that is called the *VGG119* model.

Invoice recognition refers to the process of automatically identifying an invoice within a set of documents. Ha (2017) used several supervised machine learning models, to detect the first page of an invoice from a set of documents. For this task the Logistic Regression scored best with an accuracy and F_1-score of 95%. In a study by Bouguelia, Belaïd, and Belaïd (2013) different types of financial documents, including invoices, were classified. They extended their efficient active learning method *Adaptive incremental neural gas* (*AING*) to *A2ING* in order to process both labeled and unlabeled data. A2ING is a stream-based active learning method, that is, an algorithm that updates the model incrementally with every document that becomes available from the stream. The algorithm chooses which documents are important to label by an human annotator, based on the learning results of each document it queries the class.

Predicting details of an accounting journal entry is done by, among others, Bengtsson and Jansson (2015). They tried to predict account codes with the use of classification algorithms such as Support Vector Machines with stochastic gradient descent and a Feed-Forward Neural Network. No significant improvement compared to a rule-based deterministic approach was achieved, which is possibly due to the heavy reliance on inconsistent labels in the classification algorithms. Therefore, they suggested the use of unsupervised or semi-supervised learning, and in particular the use of clustering. Similar to the case of Bengtsson and Jansson (2015), there might be some inconsistent classes present in our research, due to manual labelling. This needs to be taken into consideration.

3 Data

In this study, we consider both labeled and unlabeled financial document pages processed by the Blue10 software. Financial document pages belonging to 10 customers of Blue10 are retrieved and a random selection of financial document pages are manually assigned a class label based on their payment term. In total this is 16077 labeled financial document pages and 3500660 unlabeled financial document pages. The primary objective behind labeling a random selection of financial document pages from 10 different customers was to investigate pages with diverse layouts. This diversity arises because each vendor typically adheres to its unique document format, including the presentation of payment terms whenever they are provided. In the context of machine learning models, it is crucial to include a diverse range of pages in the training data. This ensures that the model can effectively learn from a multitude of formats, thereby avoiding any potential bias towards specific page layouts.

An example invoice page can be found in Appendix Figure 7. Invoice text is considered semi-structured data because invoices typically contain certain structured elements, such as predefined fields like invoice number and date. However, the main content, including product descriptions and customer notes, lacks a predefined format. Additionally, the content within invoices is typically not presented as a single continuous flowing text. Most invoices contain the same basic structure and information like vendor name, invoice date and total amount due. However, the specific placement and values of each field may vary across different invoices. Humans can easily identify relevant information in an invoice, such as the payment term and invoice date. The payment term of the invoice in Appendix Figure 7 is 30 days and the invoice date is 1-3-2022. Automating this process with the use of machine learning algorithms requires the extraction of numerical features.

The main source of information for the task of classifying the payment term of an invoice is the text data present in the invoice itself, the raw text data. Optical Character Recognition (hereafter OCR), can be used to convert an image or scanned document into a text document. The OCR-extraction is done by renowned Google services.

3.1 Classes

The payment term of a financial document page is not provided, necessitating manual labeling based on information in the invoice text before any (semi-)supervised machine learning algorithm can be applied. Manual labeling involves selecting the string of text that corresponds to the payment term information and converting it into a standardized format. For instance, an invoice that includes the text "We kindly ask you to succeed the payment within 7 days" and another invoice with "invoice date: 2 February 2023" and "due date: 9 February 2023" can both be labeled with a payment term of 7 days. In cases where no payment term information can be extracted from the text, the label *"Other"* is assigned. The remaining pages that are labeled with *"Other"* are pages that represent: payment reminders with no new payment due date, credit notes, an invoice where the payment is settled via direct debit, packing slips, an invoice where the payment is settled upfront, a financial document where multiple terms are included and invoices where payment is to be settled using either credit invoices from the vendor or one's own previously sent invoices to them.

The pages are labeled with their payment term, representing a number of days. The distribution of pages across these classes can be observed in Table 1. In reality, a payment term, and therefore a class, could span any number of days. However, within our labeled dataset, there are 65 distinct classes where the financial document pages can belong to. The largest class (*"Other"*) comprises roughly 56% of all pages. The class imbalance is given explicit consideration during the evaluation process.

Table 1: Payment Term Classes

Class	Amount of pages (%)	Class	Amount of pages (%)
Other	55.53%	22 days	0.03%
30 days	21.93%	62 days	0.03%
14 days	9.67%	32 days	0.03%
21 days	2.69%	42 days	0.03%
45 days	1.55%	25 days	0.02%
7 days	1.34%	35 days	0.02%
8 days	1.20%	11 days	0.02%
60 days	1.16%	90 days	0.02%
15 days	1.07%	19 days	0.02%
28 days	0.76%	24 days	0.02%
10 days	0.74%	92 days	0.02%
5 days	0.26%	26 days	0.02%
31 days	0.21%	34 days	0.01%
20 days	0.17%	33 days	0.01%
56 days	0.14%	120 days	0.01%
61 days	0.12%	39 days	0.01%
3 days	0.11%	50 days	0.01%
2 days	0.09%	36 days	0.01%
40 days	0.09%	71 days	0.01%
17 days	0.09%	84 days	0.01%
18 days	0.08%	66 days	0.01%
1 days	0.08%	65 days	0.01%
16 days	0.07%	53 days	0.01%
12 days	0.07%	74 days	0.01%
4 days	0.06%	86 days	0.01%
29 days	0.06%	64 days	0.01%
46 days	0.04%	41 days	0.01%
27 days	0.04%	59 days	0.01%
6 days	0.04%	43 days	0.01%
9 days	0.04%	23 days	0.01%
13 days	0.04%	140 days	0.01%
		44 days	0.01%

4 Methodology

This section provides an overview of all the methods used to convert the raw document text into numerical representations, to classify the payment term of a financial document page and to evaluate the performance of the developed methods.

4.1 Text to Features

Each document page is passed through an OCR system, which extracts text tokens and their positions. Problems such as nonsensical combinations of words or pieces of text due to different document formats and misinterpreting characters due to poorly scanned documents can arise and add noise. However, there have been no attempts to identify or rectify these errors. The expectation is that these errors do not significantly impact the overall analysis of the invoice text, and specifically, these errors are not expected to influence a payment term classification model.

Before the textual document data are converted into numerical input data, the data are pre-processed. The text is converted to lowercase and punctuation removal is performed. Additionally, any user-specific information such as email addresses, URLs, addresses, and numerical sequences representing phone numbers or coordinates are replaced with entity-specific tokens, such as "ADD" for addresses, to ensure anonymity. An overview regarding the categories of text replaced and the technique employed can be found in Appendix Section 7.3.4. The replacement of user-specific information with tokens helps reduce the variability of tokens.

Hereafter, the invoice text is divided into tokens in a consistent and task specific way, this is called tokenization. Tokenization is implemented using *NLTK's word_tokenize* (Loper & Bird, 2002).

Next, stop words are removed. Words that occur more frequently and which are not significant in terms of content are called stop words. Each language has its own stop words. English stop words, for example, may include the words "the", "unless" and "each". Removing these stop words from the words of the corpus reduces the input dimension and, therefore, improves scalability without losing information. Several lists of stop words are available on the internet, ranging from uni-lingual to multi-lingual lists. This research makes use of the list of stop words included in the pip package called *NLTK* (Loper & Bird, 2002). This list contains words from languages as Dutch and English, which are the most common languages in the given corpus. If the corpus still contains task-irrelevant words that appear frequently but do not carry significant information relevant to classifying the payment term of the page, these custom stop words are also removed. The removal of stop words reduces token frequency and noise in the data, potentially improving the accuracy of downstream tasks.

Last, we replace rare tokens with a special token represented as "<UNK>", where the term "UNK" is an abbreviation for unknown. This step is valuable in handling large corpora, as they tend to include a multitude of unique tokens. Tokens that appear infrequently in the corpus, falling below a predefined frequency threshold, are all mapped to the "<UNK>" token. This practice not only helps to preserve minimal information about uncommon tokens but also enables a future model to deal with out-of-vocabulary tokens when documents outside the current corpus are presented. To establish the threshold for identifying these rare tokens, we turn to Zipf's law. Zipf's law follows a power-law distribution, that is a statistical distribution that describes data where a small number of items occur frequently, while the majority of items occur infrequently. Zipf's law can describe the distribution of word frequencies in natural language, suggesting that a small number of words are extremely common, while the majority are rare (Manning, 2009). When token frequencies against their ranks are plotted on a log-log scale, we should observe a roughly straight line that starts at the top left corner of the plot and gradually slopes downward as rank increases. The closer the slope is to -1, the more closely it adheres to Zipf's law. At the beginning of the plot, you often see a steep drop, representing the most common tokens. As you move to the right along the x-axis, the line should flatten out, indicating that token frequencies decrease less rapidly. This long tail represents rare and infrequent tokens. The point where the line starts to deviate from a straight line signifies the transition from common to rare tokens. This is the point where we set the threshold. Figure 1, created by Zhang (2008), illustrates the distribution of word frequencies on a log-log scale for the words found in the novel "Ulysses" by James Joyce. The blue points on the plot represent empirical data, providing an illustration of a possible distribution. Utilizing linear regression on the log-log plot, a straight line with a slope of -1.03 can be derived. Therefore, the distribution closely adheres to Zipf's law.

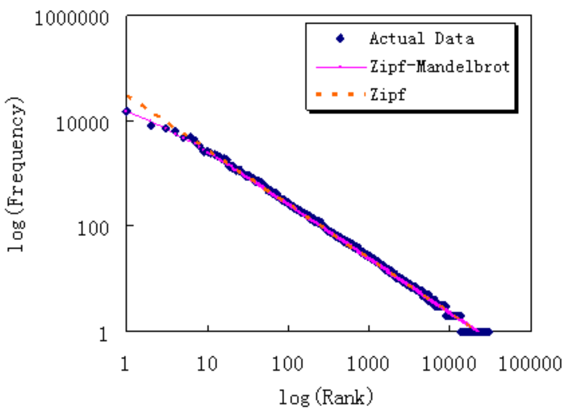


Figure 1: The distribution of word frequencies on a log-log scale in the novel "Ulysses"

In the remainder of this section, all methods that are used to convert the pre-processed textual data into feasible feature sets are described. An overview of the produced feature sets can be found in Appendix Section 7.3.3

4.1.1 BoW

The BoW representations are confined to *unigrams* and *bigrams* at the word level, where unigrams represent single words and bigrams represent sequences of two adjacent words. Lists of all the unique unigrams and bigrams present in the corpus are generated. As the sequence of adjacent words grows larger, the lists expand, demanding more computational and memory resources for processing. However, employing larger sequences of adjacent words can enhance text comprehension by capturing richer contextual information and word relationships.

The generated lists are used to create separate Document-Term matrices for unigrams and bigrams. These matrices serve as Bag-of-Words representations with TF-IDF weighting, where each unigram or bigram in every document is assigned a feature value based on its TF-IDF score. These resulting Document-Term matrices are then utilized as input datasets for the machine learning algorithms.

TF-IDF

$$TF_{ij} = \frac{f_{ij}}{max_k \ f_{kj}} \tag{3}$$

$$IDF_i = log_2 \frac{N}{n_i} \tag{4}$$

$$TF\text{ - }IDF_{ij} = TF_{ij} \times IDF_i \tag{5}$$

The formula used for TF-IDF weighting can be found in Equation 5. In this formula the term frequency, as defined in Equation 3, is multiplied with the inverse file frequency, defined in Equation 4. In the equations, N equals the number of documents in the corpus, f_{ij} is the frequency of token i in document j and n_i equals the number of documents in which token i appears. The token i with the highest TF-IDF value characterizes document j the best (Leskovec, Rajaraman, & Ullman, 2020).

SVD The dimensions of the produced Document-Term matrices are high, especially when a large sequence of tokens is used to produce features. To illustrate, the column dimension of a Document-Term matrix created with bigrams is typically higher than the column dimension of a Document-Term matrix created with unigrams. With unigrams, each word in the text is considered as a distinct feature. However, with bigrams, each feature represents a combination of two consecutive words. The SVD of a matrix can be used to derive an approximation of the the original Document-Term matrix with a significantly lower dimensionality.

Dimension reduction is performed with *Scikit-Learn's TruncatedSVD* (Pedregosa et al., 2011). *TruncatedSVD* implements a truncated randomized SVD, aiming to approximate a matrix by retaining only the top k singular values and their corresponding singular vectors.

In this implementation U, Σ and V^T are approximated with the use of the randomized algorithm developed by Halko, Martinsson, and Tropp (2010). Here, the original matrix is multiplied by a random Gaussian orthonormal matrix to obtain a smaller matrix. The random matrix, often referred to as a random projection, has entries that are drawn from a Gaussian distribution and which are then orthogonalized to form an orthonormal matrix. The dimensions of this matrix are $m \times k$, where m is the number of columns in the original matrix, and k is the desired number of singular values to be retained.

The obtained matrix, produced by the multiplication, preserves the essential properties of the original matrix while reducing its size. Then, the standard SVD algorithm is applied to this matrix to compute an approximation of the SVD. This implementation is well-suited for large matrices, as randomized techniques, such as the one applied here, generally demand fewer computational and memory resources compared to standard SVD implementations.

The resulting decomposition, provides a reduced-rank approximation of the original matrix, which can then be used to reduce the dimensions of the original matrix. In the *TruncatedSVD* implementation, $A^* = AV$ is applied to obtain a new matrix A^* in a lower dimensionality. Applying this variant of SVD to a TF-IDF weighted Document-Term matrix is known as latent semantic analysis.

The value of k is determined by the explained variance ratio, this is the summed variance of the columns after the transformation divided by the summed variance of the columns before the transformation. In this research, we aim to retain at least 90% of the original variability. Therefore, a threshold of at least 90% for the explained variance ratio is chosen. This ensures that a substantial portion of the original information is retained in the truncated representation.

Memory issues may arise when dealing with large Document-Term matrices or when the SVD computations and results become excessively large. To address this, we employ a strategy of computing the SVD on a subset of observations from the complete matrix. This subset is generated by randomly selecting a number n^* of observations, where n^* is smaller than the total number of observations, and is determined to be as large as possible while avoiding memory issues. The *TruncatedSVD* implementation is then applied to this subset.

However, in scenarios where the column dimension of the matrix is sufficiently large that even the subset, designed to avoid memory issues, consists of only a few observations (less than 10,000), producing a good approximation of the whole document-term matrix becomes challenging. In such cases, we resort to an alternative solution. Initially, we employ the *VarianceThreshold* by *Scikit-Learn* (Pedregosa et al., 2011). This method identifies and eliminates columns with low variance, thereby filtering out those that show minimal variation across documents, and are therefore less likely to offer meaningful information for machine learning models. The threshold is set using the elbow point in the plot of all sorted column variances. The elbow point is a common heuristic in mathematical optimization to choose a point where diminishing returns are no longer worth the additional cost. Figure 2 provides a visualization of the elbow point. Subsequently, a substantial subset is extracted to approximate the SVD, similar to the previous approach. This strategy helps manage the computational and memory challenges associated with large-scale matrix operations.

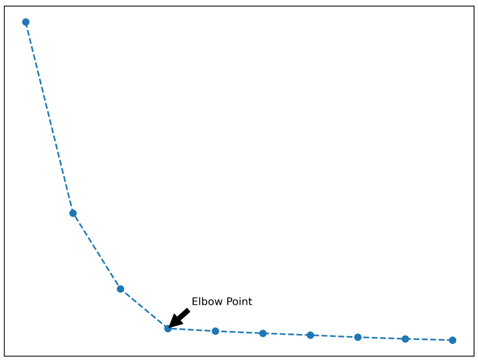


Figure 2: Elbow Point

4.1.2 Document Representations

Skip-gram and FastText models are utilized to convert words into numerical vectors, with implementation facilitated through the *Gensim* library, an open-source Python library for NLP (Rehurek & Sojka, 2011). Hyperparameters, such as the dimension of the word vector, maximum distance between words in a sentence to be considered, and the minimum word count are fine-tuned through a grid search. The grid includes various values to optimize results. Specifically, the dimension of the word vector is chosen from either 100 or 300, and the window size is set to be either 5 or 15. The minimum word count is fixed at 1. It is important to note that rare words were addressed during the pre-processing steps.

Although the CBoW and GLOVE models are mentioned in Section 2, they are not implemented. The CBoW model is an alternative to the Skip-Gram model within Word2Vec. The

GLOVE model can be relatively straightforward to implement by utilizing open-source resources that provide pre-trained GLOVE embeddings. However, implementing the GLOVE model without relying on pre-trained embeddings can be more complex and time-consuming. Therefore, other models fine-tuned specifically for the given corpus are preferred.

To derive a document representation from the word embeddings produced by the Skip-gram and FastText models, the word vectors are multiplied by the corresponding TF-IDF values of the tokens. These modified vectors are then accumulated and divided by the summed TF-IDF values. The resulting vector represents the document and serves as its feature representation (Schmidt, 2019).

Doc2Vec is a model that generates document embeddings that capture the overall semantics of the document. Doc2Vec is implemented using the *Gensim* library (Rehurek & Sojka, 2011). The training algorithm that is used is the distributed memory model. This model aims to predict a target word given its context and the document is seen as an additional input. This model requires specific hyperparameters, including the dimension of the document vector, the amount of times a word must appear in the corpus to be considered and the maximum distance between the current and predicted word within a sentence. Again, multiple values are tested in order to achieve the best possible outcome. Specifically, the dimension of the document vector is either 300 or 1000, and the window size is set to either 5 or 15. The minimum count is set to 1. The resulting vector represents the document and serves as its feature representation. Document embeddings aim to represent the overall meaning of an entire document. The choice of the embedding sizes is guided by the complexity of the information we want to capture, resulting in larger size values for document embeddings compared to those of the word embeddings.

Default values are used for all other hyperparameters. For specific values, please refer to the documentation of *Gensim* [1] [2] [3].

4.1.3 Cross-Lingual Feature Representations

The next text representations that are evaluated, are included in the method developed by Van der Goot et al. (2018), which is called bleaching text. In this method the text tokens are converted into abstract features. Then, with these abstract features a BoW representation is created, similar to the BoW representation created for N-grams. This involves creating a BoW representation with TF-IDF weighting. In this representation, the TF-IDF score serves as the feature value for each abstract feature in each document.

[1]https://radimrehurek.com/gensim/models/doc2vec.html

[2]https://radimrehurek.com/gensim/models/word2vec.html

[3]https://radimrehurek.com/gensim/models/fasttext.html

Six feature sets were created based on the bleaching text variations proposed by Van der Goot et al. (2018) by using their original code [4]. The feature sets for the bleached data variations were created based on length, punctuation usage, capital casing, and vowel/consonant-structure. Examples illustrating how text tokens are converted to abstract features in each variation are given for the six variations in Table 2. The leftmost column refers to the name of the resulting feature set. The column header includes examples of original tokens.

The sixth representation is a combination of all methods described in Table 2, the token *a* is for example converted to *01_W_W_L_V*. This feature set is indicated by *Bleach Text ALL*. The authors also proposed a frequency based representation, however, the BoW representation described in Section 4.1.1 covers the same idea, and therefore, this feature set is not created. An explanation for each feature generation method resulting in the feature sets, mentioned in the row headers of Table 2, can be found in Appendix Section 7.3.6.

Table 2: Examples for Converting Text Tokens to Abstract Features

		a	Erasmus	7	Company123@gmail.com
Bleach Text L	Length	01	07	01	020
Bleach Text C	Punctuation	W	W	W	W@W.W
Bleach Text A	Punctuation	W	W	W	WPWPW
Bleach Text S	Shape	L	ULL	D	ULLDDXLLXLL
Bleach Text V	Vowel	V	VCVCCVC	O	CVCCVCCOOOOCCVVCOCVC
Bleach Text ALL	Concatenation of all above	01_W_W_L_V	07_W_W_ULL_VCVCCVC	01_W_W_D_O	020_W@W.W_WPWPW_ULLDDXLLXLL_CVCCVCCOOOOCCVVCOCVC

Just like in the Document-Term matrices for unigrams and bigrams, sparsity and high dimensionality may occur, especially for the feature sets *Bleach Text V* and *Bleach Text ALL*. The dimension reduction techniques as described in Section 4.1.1 are applied on the created BoW representations that have more columns than labeled observations (labeled rows), in order to simultaneously reduce the dimension and retain at least 90% of the explained variance.

4.1.4 BERT

Approaches that produce numerical vector representations for sentences or paragraphs using BERT is to average the BERT output layer or to use the representation of the earlier mentioned [CLS] token. However, Reimers and Gurevych (2019) showed that this practice often yields suboptimal results, particularly when these representations are intended for tasks involving similarity measures. To address this, they developed Sentence-BERT. Sentence-BERT modifies the BERT network architecture by incorporating Siamese network structures during training. Siamese networks are designed to learn similarity between pairs of inputs. The architecture involves two identical BERT models with shared weights. During training, the Siamese network

[4] Code: https://github.com/bplank/bleaching-text

takes pairs of input sentences and learns to output similar representations for semantically similar sentences and dissimilar representations for non-similar sentences. The BERT models process pairs of sentences independently, but their parameters are updated simultaneously during training. Once trained, vector representations for input sentences are produced by adding a pooling operation to the output of BERT, such as the mean of all output vectors or the output of the [CLS]-token. By training in this manner, Sentence-BERT ensures that sentences with similar meanings or contexts are closer together in the resulting vector space. This work is the initial work of the Sentence Transformers framework, a Python framework for sentence, text and image embeddings.

Distiluse-base-multilingual-cased-v2 is a pretrained Sentence-BERT model available through the Sentence Transformers library. It operates by mapping paragraphs to a 512-dimensional dense vector space using mean pooling.

The model follows a teacher-student approach, where a teacher model produces sentence embeddings in one language. The student model aims to replicate the teacher's behavior, ensuring that comparable text across different languages are positioned closely in the vector space. To achieve this, the student model is trained on translated sentences, ensuring that the translation of each sentence is also mapped to the same vector as the original sentence (Reimers & Gurevych, 2020).

In our case, the teacher is the multilingual Universal Sentence Encoder (Yang et al., 2019). The student is a DistilBERT-based model, a compact and lighter version of BERT (Sanh, Debut, Chaumond, & Wolf, 2020).

The *distiluse-base-multilingual-cased-v2* model supports over 50 languages including Dutch and English and does differentiate between uppercase and lowercase letters. To illustrate, it differentiates between "dutch" and "Dutch". However, during the pre-processing steps, we converted all text to lowercase, enhancing the flexibility of the embeddings in relation to one another. The model can be used to encode the financial document pages. The resulting vector represents the document and serves as its feature representation.

By using a pre-trained language model, it is possible to leverage the knowledge learned from the large amounts of training data. Training data for the multilingual Universal Sentence Encoder consists of data from various sources, including question-answer pairs, translated pairs, the Stanford Natural Language Inference (SNLI) corpus and Google's translations to the SNLI corpus (Bowman, Angeli, Potts, & Manning, 2015). The *distiluse-base-multilingual-cased-v2* model uses parallel translated datasets from the OPUS website in the training process (Tiedemann & Thottingal, 2020).

It is important to note that when the pre-trained model is used, the features may not be optimized for the specific task at hand. Fine-tuning the model on a specific corpus or task may lead to better performance as the model can learn to capture the patterns in the data. Given that the other models are specifically fine-tuned on the given corpus, it can be valuable to include a pre-trained model such as *distiluse-base-multilingual-cased-v2*. Although the model is not specifically designed for large text documents, financial documents can be used as input text for the model.

4.2 Classifying Payment Term

In this section, the methods employed to generate models for classifying the payment term of an invoice, utilizing both supervised and semi-supervised learning approaches, are outlined. Initially, the best-performing combination of a supervised classifier and dataset is identified based on the F_1-score. Subsequently, this combination is integrated into a semi-supervised Self-training algorithm to explore potential performance enhancement. Additionally, the top two best-performing combinations of a supervised classifier and dataset are incorporated into a semi-supervised Co-training algorithm to further investigate potential performance improvement.

4.2.1 Supervised Machine Learning Methods

The first set of models that we consider are supervised classifiers that use labeled data to train the model. Three supervised learning algorithms are selected.

First, algorithms are chosen based on their performance in similar tasks. K-Nearest Neighbors and Multinomial Logistic Regression are selected for their performance, particularly in tasks such as invoice classification and the recognition of invoices (Tarawneh et al., 2019; Jadli & Hain, 2020; Ha, 2017).

Second, Radovanović and Ivanović (2008) stated that the most popular classifiers applied to text are, among others, K-Nearest Neighbors and Support Vector Machines. Therefore, Support Vector Classifier is implemented as well.

An overview of the implementation and the hyperparameter values that are taken into account per algorithm is presented below.

K-Nearest Neighbors K-Nearest Neighbors (hereafter KNN) is a supervised machine learning algorithm that operates on the principle of similarity, predicting the class of a new data point by considering the classes of its K nearest neighbors within the training dataset. The algorithm is non-parametric, meaning that it does not assume any specific distribution of the data. To identify the K nearest neighbors for a new data point, the algorithm uses a distance metric, such as the Euclidean or Manhattan distance. This metric calculates the distances between the new observation and all observations in the training set. Then, the class of the new observation is determined through a majority vote among its K nearest observations.

This algorithm is implemented using *Scikit-learn's KNeighborsClassifier* (Pedregosa et al., 2011). One of the most important hyperparameters that needs to be determined is the number of neighbors (*K_neighbors*). In binary classification, it is advisable to evaluate odd values for *K_-neighbors* to avoid ties. In multi-class classification, where there are more than two classes, ties are less likely to occur. If the input data has more noise, a higher value of K would be advisable. The values considered for *K_neighbors* are: [3, 33, 67, 99]. The weighting function within KNN assigns varying weights to individual neighbors. In an uniform weighting function, all neighbors are treated equally. Conversely, in a distance-based weighting function, closer neighbors have a greater impact on the prediction of the class of the new data point than farther ones. Both weighting function are examined. For calculating the distance between two data points, the Euclidean, Manhattan and Cosine distances are considered.

$$\textit{Euclidean Distance } d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

The euclidean distance is the straight-line distance between two points. This measure is best suited for continuous data and performs well when the features are measured on the same scale. Euclidean distance is sensitive to the scale of the individual features.

$$\textit{Manhattan Distance } d(p,q) = \sum_{i=1}^{n}|p_i - q_i|$$

The manhattan distance is calculated as the sum of the absolute differences between corresponding coordinates of two points. It is suitable for datasets with high dimensionality. Manhattan distance is less sensitive to outliers compared to the euclidean distance. The euclidean distance considers the square of differences, which can magnify the effect of outliers

$$\textit{Cosine Distance } d(p,q) = 1 - \frac{p \cdot q}{||p||||q||}$$

The cosine distance is calculated as 1 minus the cosine similarity between two vectors. Cosine similarity equals the cosine of the angle between two vectors. Unlike euclidean and manhattan distances, cosine distance focuses more on the direction of vectors rather than the scale of individual feature values, making it a better choice for measuring similarity in embedding spaces.

For example, in Word2Vec, words with similar meanings have embeddings that are close in the vector space. Consequently, their cosine similarity, a measure of their directional similarity, will be higher compared to embeddings of words with dissimilar meanings.

Default values are used for all other hyperparameters. For specific values, please refer to the documentation of *Scikit-Learn's KNeighborsClassifier*[5].

Multinomial Logistic Regression Multinomial Logistic regression (hereafter MLR) is designed to calculate the probabilities of an observation belonging to each class. In a classification problem, MLR assigns the observation to the class with the highest probability.

Let $y_i \in 1, ..., K$ be the class of observation i. The MLR model predicts the probability that observation i belongs to class k given the corresponding feature vector X_i ($P(y_i = k|X_i)$) as $\hat{p}_k(X_i)$ defined in Equation 7. During the training phase the cost function in Equation 6 is minimized. In this equation, W corresponds to a matrix of coefficients to be optimized, where each row vector W_k corresponds to class k. $I[y_i = k]$ is the indicator function evaluating to 1 if $y_i = k$ and 0 otherwise. $r(W)$ is the regularization term which in our case is set to the *L2* norm as shown in Equation 8. The regularization term is an additional component added to the loss function to prevent overfitting and improve the models generalization performance on unseen data. The regularization strength is controlled by C. Iterative optimization algorithms, such as stochastic gradient descent, can be used to minimize the loss function.

$$\min_{w} -C \sum_{i=1}^{n} \sum_{K=0}^{K-1} I[y_i = k] log(\hat{p}_k(X_i)) + r(W) \tag{6}$$

$$\hat{p}_k(X_i) = \frac{exp(X_i W_k + W_{0,k})}{\sum_{l=0}^{K-1} exp(X_i W_l + W_{0,l})} \tag{7}$$

$$r(W) = l_2 = \frac{1}{2}||W||_F^2 = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{K} W_{i,j}^2 \tag{8}$$

MLR is implemented using *Scikit-Learn's LogisticRegression* (Pedregosa et al., 2011). The optimization algorithms considered are *lbfgs*, *newton-cg* and *sag*. The parameter that controls the penalty strength is optimized using values on a logarithmic scale, specifically [0.1 , 1.0, 10.0]. Last, the maximum amount of iterations is set to 1,000. Default values are used for all other hyperparameters. For specific values, please refer to the documentation of *Scikit-Learn's LogisticRegression*[6].

[5]https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[6]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Support Vector Machine A Support Vector Machine (hereafter SVM) finds an optimal hyperplane that best separates data points in two different classes in the feature space. In other words, the hyperplane needs to maximize the distance between classes while minimizing classification errors. When the data is perfectly separable, the hyperplane is known as the hard margin. This can be formulated as the following primal optimization problem.

$$\begin{aligned} min_w \ \frac{1}{2}||w||^2 \\ subject\ to\ y_i(w \cdot x_i + b) \geq 1 \\ i\ =\ 1,...,n \end{aligned}$$

In this optimization problem w equals the weight vector which is normal to the hyperplane, x_i is the feature vector of a data point i, y_i is the class label and b is the bias term. The problem is a convex optimization problem that can be solved using optimization algorithms. The decision function for classifying a new data point with features x is given by $f(x) = sign(w \cdot x + b)$, where $sign(\cdot)$ is the function that assigns the class based on the side of the hyperplane the data point lies.

However, when dealing with data that is not linearly separable or when there are outliers present, the optimization problem can be modified to allow for wrongly classified data points. The hyperplane related to this case is known as the soft-margin. A slack variable ζ_i for each data point is introduced to relax the constraints of the optimization problem. The slack variable measure the degree of misclassification. C is the regularization parameter that controls the trade-off between maximizing the distance between classes and minimizing the sum of the slack variables. A larger C results in a smaller distance but fewer wrongly classified data points, while a smaller C allows for a larger distance but may lead to more wrongly classified data points. The primal optimization problem is now formulated as:

$$\begin{aligned} min_{w,b,\zeta} \ \frac{1}{2}w^T w + C\sum_{i=1}^{n}\zeta_i \\ subject\ to\ y_i(w^T\phi(x_i) + b) \geq 1 - \zeta_i \\ \zeta_i \geq 0,\ i\ =\ 1,...,n \end{aligned}$$

The following dual problem can be used to solve the primal problem more efficiently.

$$
\begin{aligned}
max_\alpha \ \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\
subject\ to\ y^T\alpha = 0 \\
0 \leq \alpha_i \leq C, i = 1, ..., n
\end{aligned}
$$

In this equation α_i are the dual coefficients, and $Q_{i,j} = y_i y_j K(x_i, x_j)$. $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel. A kernel function transforms data into higher-dimensional spaces, making it possible to find a hyperplane capable of separating non-linearly separable data. The output decision function becomes $f(x) = sign(\sum_{i\in SV} y_i\alpha_i K(x_i, x) + b)$. This function includes a summation over all support vectors x_i, all data points that lie on or within the distance boundary. These are the critical points for determining the decision boundary.

This classifier is implemented using *Scikit-Learn's SVC* (Pedregosa et al., 2011). The algorithm implements an "one-versus-one" approach for multi-class classification. In this approach, a classifier is trained for each pair of classes in the dataset. Each point is then classified according to a majority vote among the classifiers. The kernels considered are *Polynomial*, *Sigmoid*, *Linear* or *Gaussian RBF*. The definitions of these kernel are denoted in Appendix Section 7.4.1. The regularization parameter C is fine-tuned using values on a logarithmic scale, similar to the values mentioned for the parameter controlling the penalty strength in the MLR implementation, [0.1 , 1.0, 10.0]. The maximum amount of iterations is set to 100,000. Default values are used for all other hyperparameters. For specific values, please refer to the documentation of *Scikit-Learn's SVC* [7].

4.2.2 Semi-Supervised Machine Learning Methods

Our attention now turns to exploring semi-supervised algorithms to determine their potential for enhancing overall performance in classifying the payment term of a financial document page.

In our approach, we use wrapper methods which belong to a class of models that utilize both labeled and unlabeled data. The use of wrapper methods allows for direct comparison of performance results with comparable supervised classifiers. Specifically, we implement Self-training and single-view Co-training due to their simplicity and effectiveness. Both algorithms are implemented without the use of any pre-defined semi-supervised python module. The supervised classifiers discussed in Section 4.2.1 are used as parameters for the semi-supervised algorithms.

[7]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.htmlsklearn.svm.SVC

In our Self-training approach, the best performing supervised classifier is used to make predictions for the unlabeled data. The predicted observations for which the model is most certain are assigned pseudo-labels and are added to the labeled data set. This augmented data set is used to retrain the classifier. Hereafter, predictions are made for the remaining unlabeled data. This process continues until either a maximum number of initially unlabeled instances that can be added to the initially labeled data is reached, in our case this number is set to 250000, or until no new observations can be added based on the confidence threshold.

The question remains how to determine for which predicted observations the model is the most certain. Multinomial Logistic Regression calculates the probabilities of an observation belonging to each class, providing a measure of certainty regarding class membership. In the case of Multinomial Logistic Regression as the best performing supervised classifier, a prediction probability threshold is used to determine which observations to add to the labeled data set in each iteration. All observations with a prediction probability above the predefined threshold are pseudo-labeled and added to labeled data set. To explore various levels of certainty, we consider multiple thresholds, namely [0.999, 0.999999, 0.999999999]. The idea behind selecting these specific thresholds lies in the trade-off between the confidence level required for pseudo-labeling and the number of instances added to the pseudo-labeled dataset. Higher thresholds result in more confident predictions but may lead to fewer instances being pseudo-labeled. The thresholds are chosen to ensure that not for all thresholds all maximum number of instances are included in the first iteration, allowing for multiple iterations (1 or more) to be conducted.

Support Vector Machines and K-Nearest Neighbors do not directly provide prediction probabilities. For Support Vector Machines (SVM) probability estimates can be obtained using Platt-scaling. Platt-scaling requires the decision function ($f(\cdot)$) of the trained SVM classifier. Then, the parameters A and B in the following function are optimized.

$$P(y|X) = \frac{1}{(1 + exp(A \times f(X) + B)} \tag{9}$$

Platt scaling involves training a probability model based on the outputs of the SVM using a cross-entropy loss function. To avoid overfitting, this model implements five-fold cross validation.

K-Nearest Neighbors (KNN) calculates probabilities based on the classes of the K nearest neighbors. When a new data point needs to be classified, KNN identifies its K nearest neighbors in the training dataset. Once the nearest neighbors are identified, KNN can compute class probability estimates by considering the classes of these neighbors. In the case where an uniform weighting function is applied, each neighbor's class contributes equally to the probability estimate. With a distance-based weighting function, the contribution of each neighbor to the probability estimate is weighted according to its distance from the new data point.

For SVM and KNN, which are not explicitly designed for probabilistic classification, instead of using a probability threshold to determine which observations are added to the labeled data set, the top T observations based on their probability estimates are selected. For the values of T, we consider [250000, 100000, 50000]. These values are chosen to cause a small amount of iterations, which is especially convenient for SVM. The probability estimates of SVM can be computationally expensive due to the one-to-one approach applied in training the classifier and cross-validation in calculating the probability estimates.

In our single view Co-training implementation, we exploit the difference between the supervised classifiers. Initially, the top two supervised classifiers are used to make predictions for the unlabeled data. We then select the T instances for which the classifiers exhibit the highest confidence and add them to the labeled data set of the other classifier. Hereafter, the classifiers are trained again using the updated labeled datasets. These retrained classifiers are then used to predict labels for the remaining unlabeled data specific to each classifier. This process continues until a maximum number of initially unlabeled instances that can be added to the initially labeled data is reached, in our case this number is set to 250000. For the values of T we consider [125000, 50000, 25000].

4.3 Evaluation

A total of 12 different feature sets are used in the three different supervised algorithms, resulting in a total of 36 supervised models. The performance of the models is evaluated using a train and test data split. Specifically, 15% of the labeled data is randomly selected for a final test of the models. This set is an entirely independent dataset and can provide insight into how well the models generalize to unseen data.

During training, the labeled train data is divided into two datasets: a set used to train the machine learning models and a validation set. The validation set is used to tune the hyperparameters and evaluate the performance during training. The validation set consists of 20% of the train data.

Scikit-Learn's train_test_split is used to create splits that maintain the distribution of classes in each set that closely mirrors the original class distribution in the entire dataset (Pedregosa et al., 2011). This is particularly useful when dealing with imbalanced data, where some classes are underrepresented compared to others. It helps to prevent bias and ensures that the performance metrics accurately reflect a model's ability to generalize across all classes.

The hyperparameters for the supervised classification algorithms are tuned on the validation performance using grid search. A grid is created with all possible combinations of hyperparameter values, then the model performance is evaluated on the validation set for the combinations. This technique can be computationally expensive, but it is a simple and deterministic approach. Hereafter, the models using the optimal hyperparameter values are trained on the training data as well as the validation data and then evaluated with the unseen test data. This procedure prevents data-leakage, and reduces the chances of low accuracy when the chosen model is deployed.

The best-performing model, consisting of a specific combination of a dataset and an algorithm, is employed in our Self-training approach, while the top two performing supervised models are utilized in the single view Co-training approach. During each iteration of these semi-supervised learning algorithms, the validation performance and test performance are assessed. It's important to note that the unlabeled data is exclusively employed during the training phase of the semi-supervised learning algorithms. Therefore, no data split is required for the unlabeled data.

The metric used for evaluating the models is the F$_1$-Score. The F$_1$-Score is the harmonic mean between precision and recall. For multi-class classification, we calculate the Precision, Recall and F$_1$-Score for each class in an one class (in this example class A) versus the others procedure.

$$Precision_A = \frac{TP_A}{TP_A + FP_A} \tag{10}$$

$$Recall_A = \frac{TP_A}{TP_A + FN_A} \tag{11}$$

$$F_{1_A} = 2 \times \frac{Precision_A \times Recall_A}{Precision_A + Recall_A} \tag{12}$$

In an invoice processing workflow, false negatives and false positive are equally costly, since users are provided with the wrong payment term in both scenarios. Thus, no adjustments with respect to relative weights for precision and recall are necessary.

Given the large sample size and the presence of numerous classes, the decision was made not to perform resampling. Resampling, which involves artificially adjusting the distribution of the dataset, was considered undesired in this context. Preserving the original class distribution was considered more important, as the class distribution is expected to align with the real-world distribution the models will encounter.

To address the class imbalance without resorting to resampling, the weighted-average F1-Score is used. This metric considers the class imbalance by assigning weights to class-based F1-Scores relative to their actual support. By doing so, the evaluation metric provides a balanced assessment of the model's performance across all classes, accounting for the varying number of instances in each class.

$$F_1 = \frac{(\text{Observations in class A } \times \ F_{1_A}) \ + \ (\text{Observations in class B } \times \ F_{1_B}) \ + \ ...}{\text{Total observations}} \tag{13}$$

Furthermore, to exclude the contribution of the majority class on the weighted F_1-Score, a weighted F_1-Score for the classes without the inclusion of the class *"Other"* is calculated.

The accuracy of the experiments is reported as well. However, the data used in this research is heavily skewed towards the class *"Other"*. This has the consequence that the use of the accuracy metric is not insightful. When the classification works well for this dominating class, but performs poorly for the classes that are underrepresented, accuracy may still be high depending on the class imbalance.

Another metric that can be used in a multi-class classification setting is Cohen's Kappa (κ). This metric can show how well the generated model performs compared to random allocation of the classes. The definition of κ can be found in Equation 14.

$$\kappa = \frac{(p_o - p_e)}{(1 - p_e)} \tag{14}$$

In this equation p_o represents the accuracy of the generated model. p_e, on the other hand, is the accuracy achieved with random allocation. In this case Cohen's Kappa captures the predictive power of the generated model. When $\kappa > 0$, the model is said to have predicting power. Generally, if κ exceeds 0.4 the predictive power is considered good, and beyond 0.7 it is excellent.

5 Results

All data processing is executed using python 3.10. Appendix Section 7.2 provides a comprehensive list of all libraries included in the project-specific python environment.

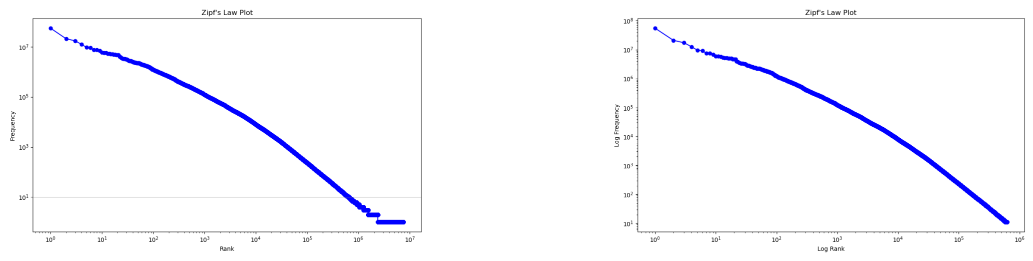
5.1 Feature Sets

5.1.1 Pre-Processing

All documents undergo the pre-processing steps outlined in Section 4.1. Converting of the the text to lowercase, punctuation removal, and anonymization are applied before forming a train-test split, as they are not directly influenced by the split and contribute to general data preparation. In contrast, stop word removal and addressing rare words take place after the split. This sequential approach is crucial because custom stop words must be identified based on the train and unlabeled data exclusively. The same principle applies to determining the threshold for defining a rare word. Custom stop words and the rare word threshold are established using the train and unlabeled data to prevent any data leakage from the test set, ensuring that the test data remain new and unseen.

Custom stop words are selected based on the train and unlabeled data, and these identified words are excluded from each document in the train, test and unlabeled datasets. The stop-word extraction process, incorporating stop words from both the NLTK package and custom stop words, results in a reduction of 0.04% of unique words. The list of chosen custom stop-words can be found in Appendix Section 7.3.2.

Additionally, replacing rare words with the UNK-token "<UNK>" leads to a further reduction of 91.87% of unique words. The threshold for this replacement is determined based on the plot presented in Figure 3. The word distribution approaches a straight line, indicative of following Zipf's law. The deviation from this line occurs when the log rank approaches 1,000,000. To manage the vocabulary size and computational complexity, a more flexible threshold is adopted. This decision aims to ensure that an adequate number of words are replaced with the "<UNK>" token, causing better model generalization. Additionally, it helps prevent overfitting to rare words in the training data, which may not generalize well to unseen data. The threshold is determined by rounding up to the nearest multiple of 10, resulting in our case to a threshold value of 10. This threshold dictates that all words with a total corpus frequency below or equal to 10 are replaced with "<UNK>". Then, this "<UNK>" token is handled as a new unique word. Furthermore, words in the test set that are not found in the corpus are also treated as unknown, and are similarly represented by the "<UNK>" token. This approach allows the models to handle unseen words during prediction. All word statistics based on the train set can be found in Appendix Section 7.3.1.



(a) Token Distribution

(b) Token Distribution After Mapping

Figure 3: Token distributions before and after mapping of rare words

5.1.2 Bag-of-Words

Due to computational and memory constraints, the Bag-of-Words representations in this research are restricted to word unigrams and bigrams. The unigram train vocabulary includes 614,038 unique words, the bigram train vocabulary includes 32,074,403 unique word-pairs. Utilizing these vocabularies, Document-Term matrices are constructed with the TF-IDF weighting procedure as described by Schmidt (2019). Initially, this yields Document-Term matrices with column dimensions of 614,038 for unigram-based representations and 32,074,403 for bigram-based representations.

To reduce training time and address memory issues arising from the high dimensionality of the document-term matrices, the number of columns is reduced. This reduction is achieved by retaining a subset of columns based on a specified threshold for column variances, using Scikit-learn's *VarianceThreshold*. This method eliminates columns with low variance, those that show minimal variation across observations, and are therefore less likely to offer meaningful information for machine learning models. The threshold for column variance is established by analyzing the sorted column variances (Appendix Figure 8). Specifically, a threshold of 0.001 is set for unigram Document-Term matrices, while a threshold of 0.005 is applied for bigram Document-Term matrices. Despite this reduction, the number of columns still exceeds the count of labeled instances in the training set for both unigram and bigram matrices. Consequently, SVD is employed as a subsequent step to further decrease the dimensions of the columns.

By ensuring a minimum explained variance retention of 90%, the unigram-based Document-Term matrix is reduced to 9,500 columns by using a SVD constructed on a subset of 250,000 pages, constituting a total reduction of 98.45%. Similarly, the bigram-based Document-Term matrix is reduced to 7,500 features using a SVD constructed on a subset of 200,000 pages, resulting in a total reduction of 99.98%. Detailed statistics for the train set can be found in Appendix Section 7.3.1.

5.1.3 Bleaching Text

The unique abstract token counts for the bleaching text feature generation methods, namely Bleach Text V, Bleach Text L, Bleach Text S, Bleach Text A, Bleach Text C, and Bleach Text ALL, are 35732, 151, 2397, 8, 22, and 36885, respectively. These tokens are utilized to construct Document-Term matrices using the TF-IDF weighting procedure. The token counts represent the column dimensions.

As previously mentioned, SVD reduction is applied when the column/feature dimension exceeds the number of labeled instances in the training set, which is the case for the Document-

Term matrices constructed for the Bleach Text V and Bleach Text ALL datasets. In these datasets, SVD reduction is applied directly, without prior column selection. The SVD's are approximated on a random selection of 750,000 labeled pages from the training set. The resulting Document-Term matrices matrices consist of 3000 columns for both data sets, resulting in a reduction of 91.60% for Bleach Text V and 91.87% for Bleach Text ALL. More detailed statistics on the bleaching text train sets can be found in Appendix Section 7.3.1.

5.1.4 Overview Data Sets

The generated data sets can be categorized into three types based on the intended use, namely train, test and unlabeled data sets. The train set includes 13665 document pages, the test set consists of 2412 document pages, and the unlabeled set contains 3500660 document pages. Each feature generation method, which converts text into numerical vectors, utilizes the train data to train a model or make decisions, and is then applied to convert the train, test and unlabeled datasets. The row dimensions, representing the number of pages in each dataset, remain consistent across all feature generation methods. An overview of the column dimensions is depicted in Table 3.

Table 3: Column Dimensions Feature Sets

Features Based On	(Optional) Hyperparameters	Number of Columns
Bag of Words: Unigrams		9500
Bag of Words: Bigrams		7500
BERT Algorithm		512
Bleach Text A Algorithm		8
Bleach Text C Algorithm		22
Bleach Text L Algorithm		151
Bleach Text S Algorithm		2397
Bleach Text ALL Algorithm		3000
Bleach Text V Algorithm		3000
Doc2Vec Algorithm	Vector length: 300, Window length: 15	300
Doc2Vec Algorithm	Vector length: 1000, Window length: 15	1000
Doc2Vec Algorithm	Vector length: 300, Window length: 5	300
Doc2Vec Algorithm	Vector length: 1000, Window length: 5	1000
Word2Vec Algorithm	Vector length: 300, Window length: 15	300
Word2Vec Algorithm	Vector length: 100, Window length: 15	100
Word2Vec Algorithm	Vector length: 300, Window length: 5	300
Word2Vec Algorithm	Vector length: 100, Window length: 5	100
FastText Algorithm	Vector length: 300, Window length: 15	300
FastText Algorithm	Vector length: 100, Window length: 15	100
FastText Algorithm	Vector length: 300, Window length: 5	300
FastText Algorithm	Vector length: 100, Window length: 5	100

5.2 Payment Term Prediction Supervised Learning

Table 4: Top 2 F_1-Scores Supervised Learning

Rank	Features	Algorithm	F_1-score
1	Bigram based	MLR algorithm using a Sag solver and penalty strength C of 10	0.89
2	Bigram based	SVM using a *RBF* kernel and penalty strength C 10	0.88

Table 4 shows the top two performing supervised payment term prediction models based on the F_1-Score calculated using the predictions on the test dataset. All other model specifications and statistics of the supervised payment term prediction models are reported in Appendix Tables 20, 21 and 22. Figure 4 provides a visualization of their performance expressed by the F_1-Score, the F_1-Score considering the exclusion of the *"Other"* class and the accuracy. The predictive power is visualized by Cohen's Kappa (κ). The F_1-Score varies between 39.5% and 89.3%. Among the three evaluated algorithms, SVM can be considered as the most stable algorithm for predicting payment terms in a set of financial document pages. This algorithm performs best for 11 out of 21 datasets and does not result in a worst performing model for each of the datasets. KNN excels for 8 out of 21 datasets but performs the worst for 3 out 21 datasets. MLR achieves the best performance in only 2 out of 21 test results and exhibits the worst performance in 18 out of 21 datasets. However, the best performing model based on the F_1-Score (89.3%) is produced using the MLR algorithm.

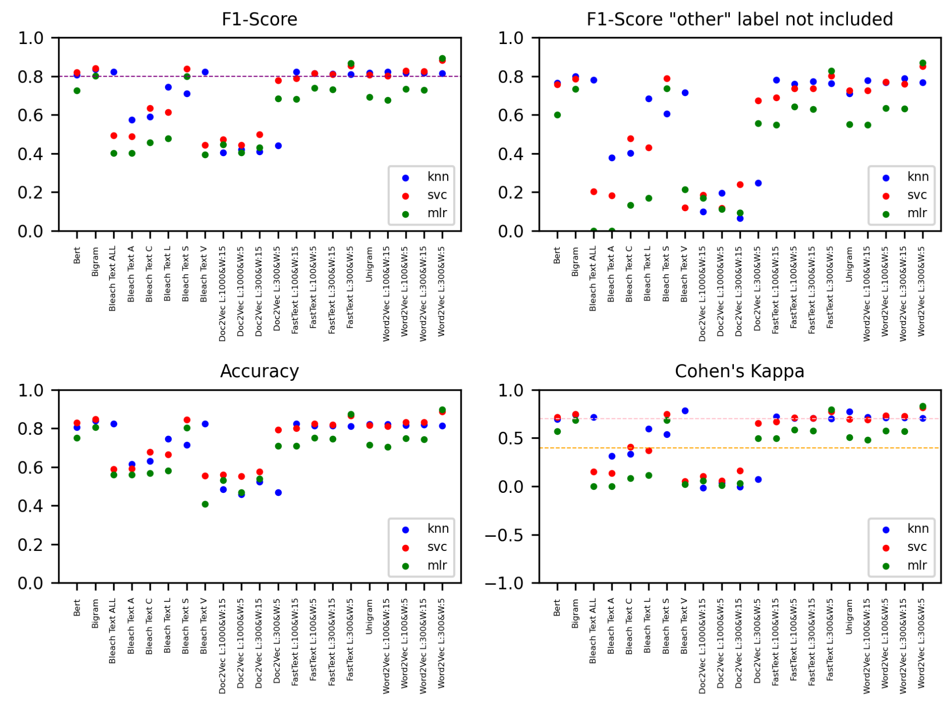


Figure 4: Test Results Supervised Learning

The MLR algorithm did not converge to a stable solution within the specified maximum number of iterations for 9 out of 21 datasets. In logistic regression, convergence refers to the algorithm's ability to find the optimal coefficients that minimize the loss function and accurately predict the payment term. The cases in which the algorithm did not converge are written in red in Appendix Tables 19 and 21.

The results of the grid search consistently indicate that the RBF kernel and a penalty strength of 10 are the optimal hyperparameters for SVM across all datasets. Regarding the KNN algorithm, it's noteworthy that using 3 neighbors is optimal for most datasets, except for those created with the bleaching text feature generation methods: Bleaching Text A and C. These datasets have relatively small column dimensions, 8 and 22 respectively, which could explain why more neighbors are optimal for them. Additionally, the majority of datasets show that a distance-based weighting function is optimal, except for the Doc2Vec dataset with a length of 300 and a window usage of 15. Cosine distance emerges as the optimal choice for the majority of datasets, particularly those utilizing embeddings like Word2Vec and FastText. This preference is driven by the nature of the embeddings and cosine distance, which measures similarity based on the direction of vectors rather than their magnitudes. In high-dimensional embedding spaces, where vectors represent connections between words or phrases based on their meanings, cosine distance effectively captures resemblance in meaning by focusing on vector direction.

The bigram based dataset is the best performing set, delivering good results across all three algorithms. It achieves a F_1-Score surpassing 83% for all three algorithms. Notably, when paired with this set, MLR and SVM stand out as the two leading performers based on the F_1-Score. The feature set derived from bigrams appears to be more suitable than the feature set based on unigrams. This is evident across all algorithms, where the bigram based dataset consistently yields a higher F_1-Score.

The feature sets created with the Doc2Vec algorithm result in inferior performance compared to the other feature sets constructed at the word level. This implies that, in terms of predictive accuracy or effectiveness for predicting payment terms, the embeddings derived from Doc2Vec at the document level are not as successful as those generated from alternative methods that focus on individual words.

Inline with Van der Goot et al. (2018), the concatenated bleaching feature set outperforms generic lexical features when used as data in all three the algorithms. However, this improvement is marginal when contrasted with lexical features extracted from vowel structures.

The hyperparameters in the feature generation methods reveal specific trends. For the

Doc2Vec algorithm, a smaller window size (5 versus 15) is found to be more suitable. Conversely, for the Word2Vec algorithm, a larger window size (15 versus 5) proves to be more effective. Interestingly, for the FastText algorithm, a smaller vector length (100 versus 300) yields better performance based on the F_1-Score. However, the differences in F_1-Scores are minimal.

The F_1-Score is a metric that balances precision and recall and considers the class distribution in the data. When the largest class *"Other"*, which is substantially larger than the other classes, is excluded from evaluation, the F_1-Score decreases. Tables 23, 24 and 25 show the absolute difference, as well as the difference expressed as a percentage of the overall F_1-Score. Generally the negative impact of the exclusion of the "*Other*" class on the F_1-Score increases when the the F_1-Score decreases. This suggests that the model maintains robust performance on smaller classes in the good performing experiments. Meaning that the model's ability to correctly classify observations in the smaller classes is not compromised by the dominance of the largest class. The negative impact on the scores produced by the MLR algorithm are the largest.

The predictive power for the majority of models, 42 out of 63, can be considered good as κ exceeds 0.4. The predictive power for 17 models is considered excellent, given that κ is larger than 0.7. The combination of the MLR algorithm with the bigram dataset, which yielded the highest F_1-Score, also has the highest predictive power κ (0.87).

5.3 Payment Term Prediction Semi-Supervised Learning

In our semi-supervised learning implementations, we aim to enhance the performance of the top-performing supervised model based on the F_1-Score, specifically the MLR algorithm coupled with the bigram dataset.

5.3.1 Self-Training

In our Self-training approach, the optimal hyperparameters obtained from the supervised counterpart are adopted in every iteration of the Self-training algorithm. In particular, the MLR algorithm is set to use the *sag* algorithm in every iteration, and the penalty strength is specified as *10.* We assess the efficacy of three distinct thresholds for selecting unlabeled instances that can be pseudo-labeled, namely 0.999, 0.999999 and 0.999999999. The results of each iteration under these thresholds in the Self-training algorithm are illustrated in Figure 5.

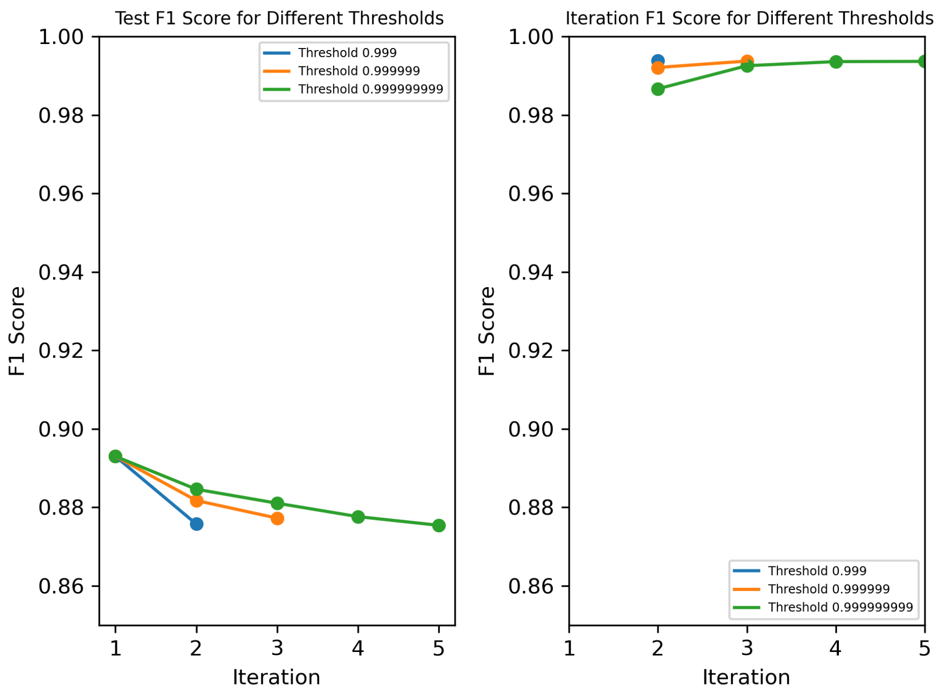


Figure 5: Results Self-Training

The F_1-Score calculated using the predictions on the test dataset is displayed in the left sub-figure, while the right sub-figure presents the F_1-Score calculated using the predictions on the validation dataset. Each line within the sub-figures corresponds to a distinct threshold, and every dot represents an iteration. Specifically, when the threshold is set to 0.999, a single iteration is executed; for a threshold of 0.999999, two iterations are conducted, and for a threshold of 0.999999999, four iterations take place.

Observing the right sub-figure, it is evident that the F_1-Scores produced with the predictions on the validation dataset are consistently higher than the final F_1-Score obtained with supervised learning (89.3%). In contrast, the F_1-Scores produced with the predictions on the test dataset exhibit a decrease with each iteration. The first F_1-Score in this plot represents the final result of the supervised algorithm, followed by a subsequent decline in F_1-Score per iteration for each of the thresholds.

With a threshold of 0.999, the final F_1-Score is 88.5%. Increasing the threshold to 0.999999 results in a slightly higher final F_1-Score of 88.6%, while a threshold of 0.999999999 yields a final F_1-Score of 88.5% when the stopping condition of adding 250,000 initially unlabeled instances is met. Notably, there are no convergence issues after a single iteration for each threshold. Furthermore, the stricter the threshold, the less noticeable the decline in each subsequent iteration until reaching the stopping criteria. For more detailed results, please consult Appendix Tables 26 and 27.

5.3.2 Co-Training

Co-training is implemented using the top two performing-models based on the F$_1$-Score as showed by Table 4. Specifically, the MLR algorithm and SVM algorithm, when coupled with the Bigram dataset. The optimal hyperparameters obtained from the supervised counterparts are applied in every iteration of the Co-training algorithm. In particular, the MLR algorithm is set to use the *sag* algorithm, the SVM algorithm uses the *RBF* kernel, and the penalty strength for both algorithms is specified as *10*.

We assess the efficacy of three distinct thresholds for selecting unlabeled instances that can be pseudo-labeled per underlying algorithm to refine the predictions of the other algorithm, namely 125000, 50000 and 25000. The results of each iteration under these thresholds in the semi-supervised learning algorithm are illustrated in Figure 6.

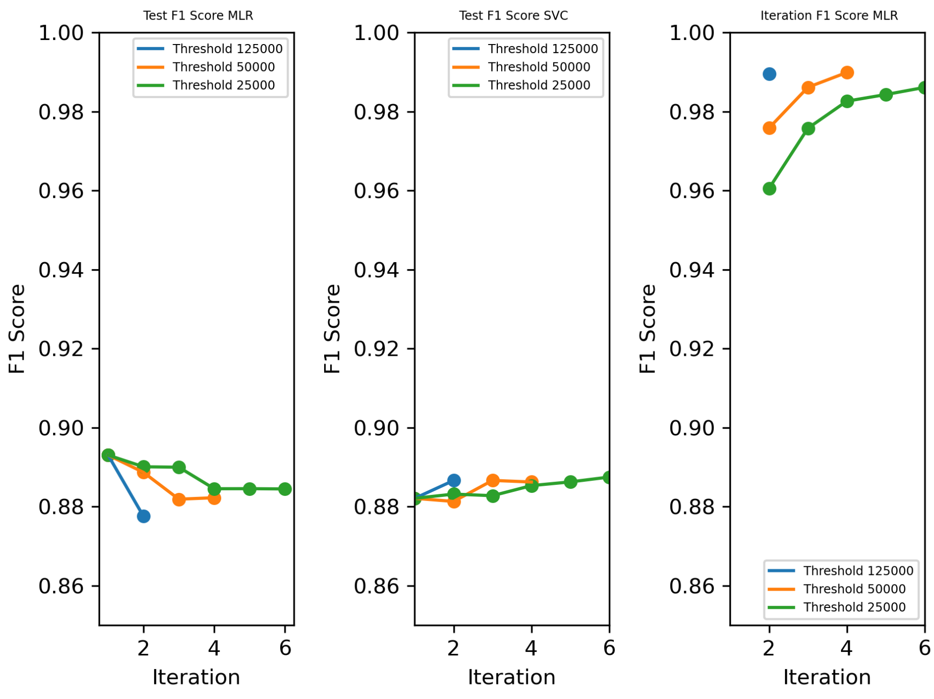


Figure 6: Results Semi-Supervised Learning

In the left sub-figure, the F$_1$-Scores are displayed based on predictions made by the MLR algorithm on the test dataset. The middle plot illustrates the F$_1$-Scores derived from predictions made by the SVM algorithm on the same test dataset. In the right sub-figure, the F$_1$-Scores are presented for the MLR algorithm on the validation set. Each line within these sub-figures corresponds to a distinct threshold. Specifically, one iteration occurs when the threshold is set to 125000, three iterations are performed for a threshold of 50000, and six iterations are executed for a threshold of 25000.

Observing the right sub-figure, it is evident that the F_1-Scores produced with the predictions on the validation dataset are again consistently higher than the final F_1-Score obtained with supervised learning (89.3%). Conversely, the F_1-Scores produced with the predictions on the test dataset using the MLR algorithm exhibit an overall decrease. The first test F_1-Score represents the final result of the supervised algorithm, followed by an overall decline in F_1-Score. With a threshold of 125000, the final F_1-Score is 87.8%. The more stricter threshold of 50000 results in a slightly higher final F_1-Score of 88.2%, while a threshold of 25000 results in the least decline with a final F_1-Score of 88.4% when the stopping condition of adding 250,000 initially unlabeled instances is met.

Notably, the F_1-Score obtained from predictions on the test dataset using the SVM algorithm exhibits an overall increasing trend. With a threshold of 125000, the final F_1-Score reaches 88.7%. Employing a stricter threshold of 50000 results in a slightly lower final F_1-Score of 88.6%, while a threshold of 25000 yields a final F_1-Score of 88.7% upon fulfilling the stopping condition of adding 250,000 initially unlabeled instances. Although there is an observed improvement in F_1-scores with the SVM algorithm, all final scores remain below those achieved by the top-performing model. Specifically, the MLR algorithm coupled with the bigram dataset attained a F_1-Score of 89.3%. For more detailed results, please refer to Table 28, 29 and 30 in the Appendix.

6 Discussion and Conclusion

6.1 Discussing Research Questions

6.1.1 Feature Sets

The first research question aims to determine the most effective procedure for converting raw text data into a numerical representation suitable for classifying the payment term of an invoice using supervised classification models such as K-Nearest Neighbors, Multinomial Logistic Regression, or Support Vector Machines, with performance measured by the F_1-Score. The findings indicate that the bigram-based feature set consistently outperforms other representation types across all three supervised algorithms, emerging as the top-performing dataset. Additionally, the unigram and Bleach Text ALL representations also demonstrate suitability for representing document pages, as measured by their performance.

Bigrams capture sequential dependencies between words, allowing the models to better understand the context and meaning of the text. This improvement in performance compared to using unigrams or abstract tokens based on single words suggests that the task of predicting the payment term benefits from not only analyzing individual words but also their interactions. This shows the importance of considering semantic relationships in invoice data processing, as it enhances the model's ability to extract meaningful information from the text.

The varying performance of abstract tokens and not one of the datasets being the top-performing dataset, suggests that the task at hand may not necessitate or benefit from the additional linguistic diversity offered by the abstract tokens. This could imply that either the data or the task of predicting the payment term does not inherently involve multi-lingual elements, leading to a situation where leveraging abstract tokens does not provide a significant advantage. Alternatively, it could indicate that the multilingual representation does not offer distinct benefits over the uni-lingual representation for our specific task.

In contrast to the findings of Le and Mikolov (2014), the document embeddings do not outperform the Bag-of-Words representations. In fact, they perform the worst. Moreover, embeddings constructed at the word level outperform embeddings at the document level. This discrepancy suggests that, in this research, features derived from individual words are more effective predictors than those obtained at the document level using Doc2Vec. Possible explanations for this performance difference could involve challenges in capturing nuanced document-level semantics caused by the specific nature of invoice data or limitations in the training process that impact the quality of document embeddings. Further, no apparent difference in F_1-Score behavior is observed between the Word2Vec and FastText algorithms.

6.1.2 Supervised Learning

The second research question focuses on determining the most suitable supervised machine learning algorithm among K-Nearest Neighbors (KNN), Multinomial Logistic Regression (MLR), and Support Vector Machines (SVM) for classifying the payment term of an invoice based on the F_1-Score performance. KNN is advantageous due to its non-parametric nature, resulting in faster computation time compared to MLR and SVM, which require multiple iterations to find optimal models. SVM, in a multi-class setting, involves creating multiple models in each iteration, making it slower to train.

MLR, when combined with the bigram dataset, shows the best performance according to the F_1-Score. However, when compared with the other two algorithms, MLR performs the worst for 18 out of 21 datasets and is the best for only 2 out of 21. On the other hand, SVM performs best for 11 out of 21 datasets and does not result in the worst performance for any of the datasets.

Regardless of the dataset, SVM consistently utilizes the RBF kernel with a penalty strength of 10, while KNN tends to perform well with 3 neighbors, using a distance-based weighting function and a cosine distance measure.

It's important to note that the accuracy of predictions and test results heavily depends on the quality of the labeled data used for training the models and the algorithm's ability to generalize to unseen data. Consequently, while the MLR algorithm yields the top-performing model, the KNN algorithm with 3 neighbors, a distance-based weighting function, and cosine distance is recommended when rapid training is prioritized. On the other hand, in scenarios where training time is not a concern, employing a SVM algorithm with a RBF kernel and penalty strength of 10 is suggested.

6.1.3 Semi-Supervised Learning

The final research question examines whether semi-supervised learning can improve the performance compared to that of the supervised models. The results indicate that the best-performing supervised learner consistently outperforms the semi-supervised counterparts. Both Self-training and Co-training iterations show a decrease in performance with the use of the Multinomial Logistic Regression algorithm. Although the Support Vector Machine algorithm exhibits an increasing trend in performance across iterations, it fails to surpass the top performance achieved by the supervised model. Finally, the validation results consistently outperform the test results and show an increasing trend, possibly indicating overfitting to the training data and the *"Other"* class caused by the natural imbalance in the class distribution.

6.2 Limitations and Future Research

Despite excellent results, several limitations and future improvements can be identified. In our research, the payment term is categorized into a finite number of classes (65), whereas in real-life scenarios, payment terms can span any number of days. To address the challenge of handling a potentially infinite number of payment term classes, a two-stage approach can be adopted.

In the first stage, the problem is treated as a binary classification task. The objective here is to determine whether an invoice falls under the class *"Other"*. In other words, it is about identifying invoices where the payment term is not among the classes that represents a number of days greater than or equal to one in our dataset.

For invoices that do not belong to the *"Other"* class in the first stage, the second stage of classification is employed. Here, an algorithm, that is not bounded to classes, such as regression algorithms, is utilized since the payment term can encompass any numerical value between 1 and infinity. The (regression) model predicts a numerical value, representing the payment term, based on the independent variables. To handle the practicality of payment terms, which are typically discrete and rounded, the predicted numerical value is rounded to the nearest whole number. In cases where the predicted value is below one, it's rounded up to one. An alternative is choosing a model or an adapted version of a model that assume non-negativity. For example, an (adapted) implementation of a survival analysis model can be used. Despite using regression or other non-classification models, this approach can still be evaluated the same as a classification problem with an infinite number of classes, considering the rounding applied to the predicted values. To assess the performance of both stages, similar evaluation metrics as used in this research can be employed. The final evaluation metric is derived from classes obtained by combining both stages, providing an overall measure of the model's effectiveness.

The extensive dataset provided by Blue10 has proven extremely valuable, but it also forced modelling decisions that may have hindered performance. For instance, due to computational constraints, we were unable to thoroughly explore character N-grams or implement a full SVD. Furthermore, flexible thresholds that do not require a lot of iterations in the semi-supervised learning algorithms are used. In reality, more iterations, caused by using a stricter threshold, can be beneficial.

Additionally, future research could explore the potential benefits of rectifying OCR errors on model performance. Moreover, investigating alternative embedding or classification algorithms may lead to performance improvements. Finally, incorporating additional features that capture information present on an invoice, such as vendor details, total amount, invoice date, etc. and including an invoice's meta-data, could enhance model performance. While the features utilized in this research are based solely on the textual content before invoice processing, the inclusion of features like vendor information may offer valuable insights, potentially correlated with payment terms. However, it's essential to consider the accuracy of these additional features, as their quality directly influences model performance. Future research can investigate whether it is beneficial to add these features and try to determine which features should be added.

6.3 Conclusion

In conclusion, our research demonstrates that the payment term of an invoice can be effectively classified into a finite number of classes using a feature set based on bigrams, which is dimensionality reduced using SVD, and employing a MLR algorithm configured with the *sag* algorithm and a regularization parameter strength of *10*. The payment term of unseen invoice can be predicted to a satisfying extent, as verified by the models' high F1-scores (89.3%), accuracy (89.8%) and predictive power (83.4%).

The top performing supervised model outperforms the semi-supervised algorithms and offers faster training times due to the restriction to one iteration and a smaller amount of data available for training.

This paper has extended literature by solely focusing on predicting the payment term and including semi-supervised algorithms in the process of finding an optimal model. The findings not only contribute to the body of academic literature, but are also valuable for business applications.

While real-life scenarios may involve an infinite number of payment term classes, our model currently misclassifies such cases as either the class *"Other"* or any another numbered payment term class. However, given that the most common classes are included in our dataset, the current approach suffices for practical purposes.

Overall, the findings presented in this paper, together with proposed directions for further research, are promising for completing the goal of developing an optimal method for classifying the payment term of an invoice.

References

Adhikari, A., Ram, A., Tang, R., & Lin, J. (2019). Docbert: Bert for document classification. *arXiv preprint arXiv:1904.08398*.

Albright, R. (2004). Taming text with the svd. *SAS Institute Inc.*

Appel, A. P., Oliveira, V., Lima, B., Malfatti, G. L., de Santana, V. F., & de Paula, R. (2019). Optimize cash collection: Use machine learning to predicting invoice payment. *arXiv preprint arXiv:1912.10828*.

Aslan, E., Karakaya, T., Unver, E., & Akgül, Y. S. (2016). A part based modeling approach for invoice parsing. In *Visigrapp (3: Visapp)* (pp. 392–399).

Bengtsson, H., & Jansson, J. (2015). Using classification algorithms for smart suggestions in accounting systems. *unpublished thesis*.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on computational learning theory* (pp. 92–100).

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). *Enriching word vectors with subword information.*

Bouguelia, M.-R., Belaïd, Y., & Belaïd, A. (2013). A stream-based semi-supervised active learning approach for document classification. In *2013 12th international conference on document analysis and recognition* (pp. 611–615).

Bowman, S. R., Angeli, G., Potts, C., & Manning, C. D. (2015, September). A large annotated corpus for learning natural language inference. In L. Màrquez, C. Callison-Burch, & J. Su (Eds.), *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 632–642). Lisbon, Portugal: Association for Computational Linguistics. Retrieved from https://aclanthology.org/D15-1075 doi: 10.18653/v1/D15-1075

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, *39*(1), 1–22.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dharma, E. M., Gaol, F. L., Warnars, H., & Soewito, B. (2022). The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification. *Journal of Theoretical and Applied Information Technology*, *100*(2), 31.

Esser, D., Schuster, D., Muthmann, K., & Schill, A. (2014). Few-exemplar information extraction for business documents. In *Iceis (1)* (pp. 293–298).

Frey, C. B., & Osborne, M. A. (2017). The future of employment: How susceptible are jobs to computerisation? *Technological forecasting and social change*, *114*, 254–280.

Ha, H. T. (2017). Recognition of invoices from scanned documents. In *Raslan* (pp. 71–78).

Ha, H. T., & Horák, A. (2022). Information extraction from scanned invoice images using text analysis and layout features. *Signal Processing: Image Communication*, *102*, 116601.

Halko, N., Martinsson, P.-G., & Tropp, J. A. (2010). *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.*

Hamza, H., Belaïd, Y., Belaïd, A., & Chaudhuri, B. B. (2008). Incremental classification of invoice documents. In *2008 19th international conference on pattern recognition* (pp. 1–4).

Jadli, A., & Hain, M. (2020). Automatic document classification using deep feature selection and knowledge transfer. In *2020 1st international conference on innovative research in applied science, engineering and technology (iraset)* (pp. 1–5).

Lau, J. H., & Baldwin, T. (2016). An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*.

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188–1196).

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2020). *Mining of massive data sets.* Cambridge university press.

Liu, W., Zhang, Y., & Wan, B. (2016). Unstructured document recognition on business invoice.

Machine Learning, Stanford iTunes University, Stanford, CA, USA, Technical report.

Loper, E., & Bird, S. (2002). *Nltk: The natural language toolkit.* arXiv. Retrieved from `https://arxiv.org/abs/cs/0205028` doi: 10.48550/ARXIV.CS/0205028

Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, *1*(4), 309–317.

Majumder, B. P., Potti, N., Tata, S., Wendt, J. B., Zhao, Q., & Najork, M. (2020). Representation learning for information extraction from form-like documents. In *proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 6495–6504).

Manning, C. D. (2009). *An introduction to information retrieval.* Cambridge university press.

Meyer, C. D. (2000). *Matrix analysis and applied linear algebra* (Vol. 71). Siam.

Mikolov, T. (2013, Oct). *De-obfuscated python + question.* Google. Retrieved from `https://groups.google.com/g/word2vec-toolkit/c/NLvYXU99cAM/m/E5ld8LcDxlAJ`

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, *26*.

Palm, R. B., Winther, O., & Laws, F. (2017). Cloudscan-a configuration-free invoice analysis system using recurrent neural networks. In *2017 14th iapr international conference on document analysis and recognition (icdar)* (Vol. 1, pp. 406–413).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 1532–1543).

Plank, B. (2017). All-in-1: Short text classification with one model for all languages. *arXiv preprint arXiv:1710.09589*.

Provost, F., & Domingos, P. (2003). Tree induction for probability-based ranking. *Machine learning*, *52*, 199–215.

Radovanović, M., & Ivanović, M. (2008). Text mining: Approaches and applications. *Novi Sad J. Math*, *38*(3), 227–234.

Rehurek, R., & Sojka, P. (2011). Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, *3*(2).

Reimers, N., & Gurevych, I. (2019). *Sentence-bert: Sentence embeddings using siamese bert-networks.*

Reimers, N., & Gurevych, I. (2020, 04). Making monolingual sentence embeddings multilingual using knowledge distillation. *arXiv preprint arXiv:2004.09813*. Retrieved from `http://arxiv.org/abs/2004.09813`

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). *Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.*

Schmidt, C. W. (2019). Improving a tf-idf weighted document vector embedding. *arXiv preprint arXiv:1902.09875*.

Schuster, D., Muthmann, K., Esser, D., Schill, A., Berger, M., Weidling, C., ... Hofmeier, A. (2013). Intellix–end-user trained information extraction for document archiving. In *2013 12th international conference on document analysis and recognition* (pp. 101–105).

Smith, S. L., Turban, D. H., Hamblin, S., & Hammerla, N. Y. (2017). Offline bilingual word vectors, orthogonal transformations and the inverted softmax. *arXiv preprint arXiv:1702.03859*.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, *28*(1), 11–21.

Tang, P., Qiu, W., Huang, Z., Chen, S., Yan, M., Lian, H., & Li, Z. (2020). Anomaly detection in electronic invoice systems based on machine learning. *Information Sciences*, *535*, 172–

186.

Tarawneh, A. S., Hassanat, A. B., Chetverikov, D., Lendak, I., & Verma, C. (2019). Invoice classification using deep features and machine learning techniques. In *2019 ieee jordan international joint conference on electrical engineering and information technology (jeeit)* (pp. 855–859).

Tiedemann, J., & Thottingal, S. (2020, November). OPUS-MT – building open translation services for the world. In *Proceedings of the 22nd annual conference of the european association for machine translation* (pp. 479–480). Lisboa, Portugal: European Association for Machine Translation. Retrieved from `https://aclanthology.org/2020.eamt-1.61`

Triguero, I., González, S., Moyano, J. M., García López, S., Alcalá Fernández, J., Luengo Martín, J., ... others (2017). Keel 3.0: an open source software for multi-stage analysis in data mining.

Van der Goot, R., Ljubešić, N., Matroos, I., Nissim, M., & Plank, B. (2018). Bleaching text: Abstract features for cross-lingual gender prediction. *arXiv preprint arXiv:1805.03122*.

Van Engelen, J. E., & Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine learning*, *109*(2), 373–440.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wang, W., & Zhou, Z.-H. (2010). A new analysis of co-training. In *Icml* (Vol. 2, p. 3).

Yang, Y., Cer, D., Ahmad, A., Guo, M., Law, J., Constant, N., ... Kurzweil, R. (2019). *Multilingual universal sentence encoder for semantic retrieval.*

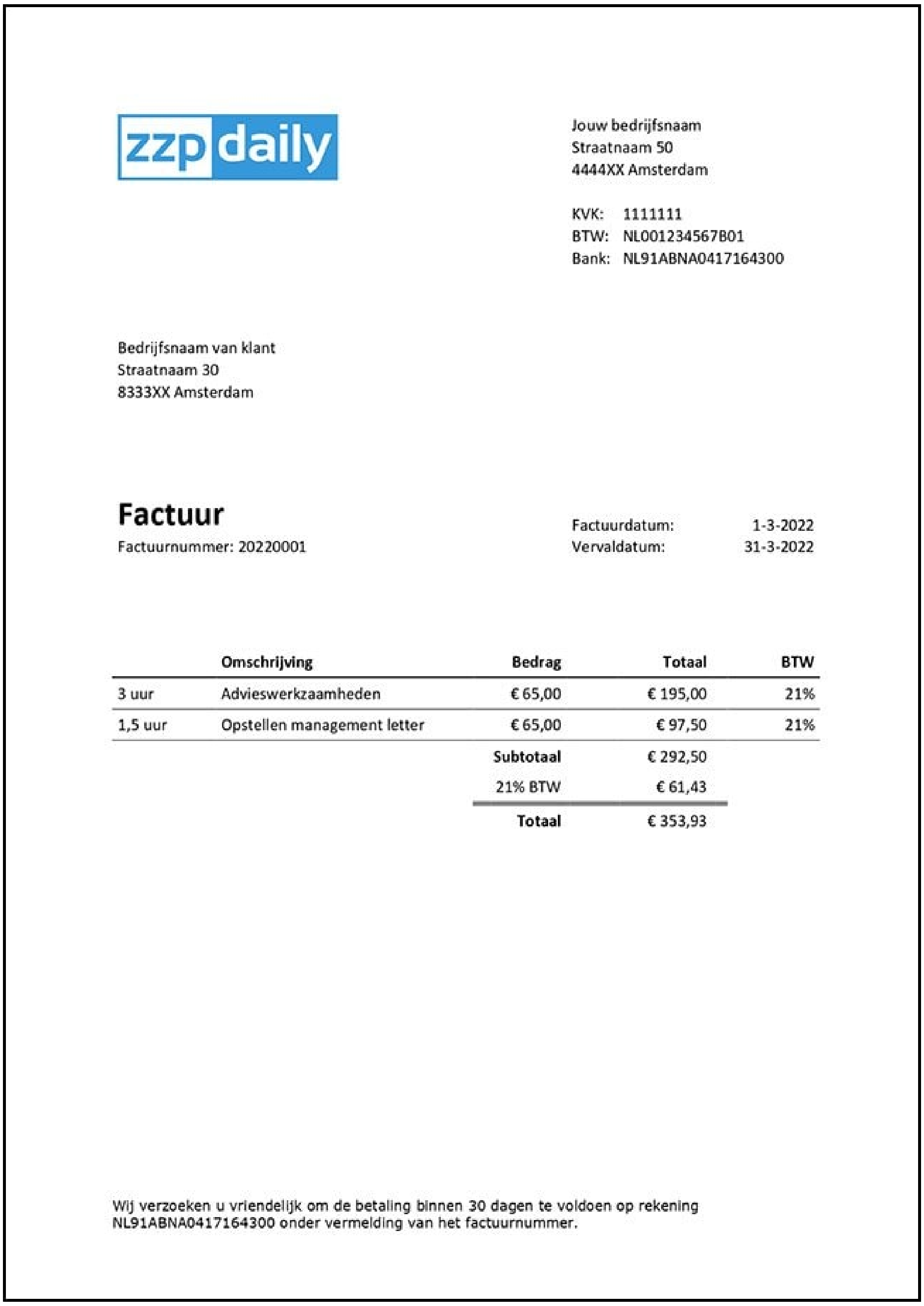
Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics* (pp. 189–196).

Zhang, H. (2008). Exploring regularity in source code: Software science and zipf's law. In *2008 15th working conference on reverse engineering* (pp. 101–110).

Zhao, X., Niu, E., Wu, Z., & Wang, X. (2019). Cutie: Learning to understand documents with convolutional universal text information extractor. *arXiv preprint arXiv:1903.12363*.

7 Appendices

7.1 Example Invoice



zzp daily

Jouw bedrijfsnaam
Straatnaam 50
4444XX Amsterdam

KVK:	1111111
BTW:	NL001234567B01
Bank:	NL91ABNA0417164300

Bedrijfsnaam van klant
Straatnaam 30
8333XX Amsterdam

Factuur
Factuurnummer: 20220001

Factuurdatum:	1-3-2022
Vervaldatum:	31-3-2022

	Omschrijving	Bedrag	Totaal	BTW
3 uur	Advieswerkzaamheden	€ 65,00	€ 195,00	21%
1,5 uur	Opstellen management letter	€ 65,00	€ 97,50	21%
		Subtotaal	€ 292,50	
		21% BTW	€ 61,43	
		Totaal	€ 353,93	

Wij verzoeken u vriendelijk om de betaling binnen 30 dagen te voldoen op rekening NL91ABNA0417164300 onder vermelding van het factuurnummer.

Figure 7: Example of an invoice

7.2 Environment Packages

Table 5: Python Modules

Package	Version	Package	Version	Package	Version	Package	Version
_tflow_select	2.3.0	google-auth	2.6.0	mkl-service	2.4.0	requests-oauthlib	1.3.0
abseil-cpp	20211102	google-auth-oauthlib	0.4.4	mkl_fft	1.3.6	responses	0.13.3
absl-py	1.4.0	google-pasta	0.2.0	mkl_random	1.2.2	rsa	4.7.2
aiohttp	3.8.3	grpcio	1.42.0	mpmath	1.3.0	sacremoses	0.0.43
aiosignal	1.2.0	gst-plugins-base	1.18.5	msgpack-python	1.0.3	scikit-learn	1.3.0
appdirs	1.4.4	gstreamer	1.18.5	multidict	6.0.2	scipy	1.10.1
astunparse	1.6.3	h5py	3.7.0	multipledispatch	0.6.0	sentence-transformers	2.2.2
async-timeout	4.0.2	hdf5	1.10.6	multiprocess	0.70.14	sentencepiece	0.1.99
attrs	22.1.0	heapdict	1.0.1	munkres	1.1.4	setuptools	68.0.0
aws-c-common	0.6.8	huggingface_hub	0.15.1	networkx	3.1	sip	6.6.2
aws-c-event-stream	0.1.6	icc_rt	2022.1.0	ninja	1.10.2	six	1.16.0
aws-checksums	0.1.11	icu	58.2	ninja-base	1.10.2	smart_open	5.2.1
azure-common	1.1.28	idna	3.4	nltk	3.8.1	snappy	1.1.9
azure-storage-blob	2.1.0	intel-openmp	2023.1.0	numba	0.57.0	sortedcontainers	2.4.0
azure-storage-common	2.1.0	jinja2	3.1.2	numexpr	2.8.4	sqlite	3.41.2
blas	1	joblib	1.2.0	numpy	1.24.3	sympy	1.11.1
blinker	1.4	jpeg	9e	numpy-base	1.24.3	tbb	2021.8.0
bokeh	3.2.1	keras	2.10.0	oauthlib	3.2.2	tblib	1.7.0
boost-cpp	1.73.0	keras-preprocessing	1.1.2	openssl	1.1.1v	tensorboard	2.10.0
bottleneck	1.3.5	kiwisolver	1.4.4	opt_einsum	3.3.0	tensorboard-data-server	0.6.1
brotli	1.0.9	krb5	1.19.4	orc	1.7.4	tensorboard-plugin-wit	1.8.1
brotli-bin	1.0.9	lerc	3	packaging	23	tensorflow	2.10.0
brotlipy	0.7.0	libboost	1.73.0	pandas	1.5.3	tensorflow-base	2.10.0
bzip2	1.0.8	libbrotlicommon	1.0.9	partd	1.2.0	tensorflow-estimator	2.10.0
c-ares	1.19.0	libbrotlidec	1.0.9	pcre	8.45	termcolor	2.1.0
ca-certificates	2023.7.22	libbrotlienc	1.0.9	pillow	9.4.0	threadpoolctl	2.2.0
cachetools	4.2.2	libclang	14.0.6	pip	23.2.1	tk	8.6.12
certifi	2023.7.22	libclang13	14.0.6	ply	3.11	tokenizers	0.13.2
cffi	1.15.1	libcurl	8.1.1	pooch	1.4.0	toml	0.10.2
charset-normalizer	2.0.4	libdeflate	1.17	protobuf	3.20.3	toolz	0.12.0
click	8.0.4	libffi	3.4.4	psutil	5.9.0	torchvision	0.15.2
cloudpickle	2.2.1	libiconv	1.16	pyasn1	0.4.8	tornado	6.3.2
colorama	0.4.6	libllvm14	14.0.6	pyasn1-modules	0.2.8	tqdm	4.65.0
contourpy	1.0.5	libogg	1.3.5	pycparser	2.21	transformers	4.24.0
cryptography	41.0.2	libpng	1.6.39	pyjwt	2.4.0	typing-extensions	4.7.1
cycler	0.11.0	libprotobuf	3.20.3	pyopenssl	23.2.0	typing_extensions	4.7.1
cytoolz	0.12.0	libssh2	1.10.0	pyparsing	3.0.9	tzdata	2023c
dask	2022.2.1	libtiff	4.5.0	pyqt	5.15.7	urllib3	1.26.16
dask-core	2022.2.1	libuv	1.44.2	pyqt5-sip	12.11.0	utf8proc	2.6.1
dask-glm	0.2.0	libvorbis	1.3.7	pysocks	1.7.1	vc	14.2
dask-ml	2023.3.24	libwebp	1.2.4	python	3.10.12	vs2015_runtime	14.27.29016
dill	0.3.6	libwebp-base	1.2.4	python-dateutil	2.8.2	werkzeug	2.2.3
distributed	2022.2.1	libxml2	2.10.3	python-flatbuffers	2	wheel	0.38.4
filelock	3.9.0	libxslt	1.1.37	python-lmdb	1.4.1	win_inet_pton	1.1.0
flatbuffers	2.0.0	llvmlite	0.40.0	python-xxhash	2.0.2	wrapt	1.14.1
fonttools	4.25.0	locket	1.0.0	pytorch	2.0.1	xxhash	0.8.0
freetype	2.12.1	lz4-c	1.9.4	pytz	2022.7	xyzservices	2022.9.0
frozenlist	1.3.3	markdown	3.4.1	pyyaml	6	xz	5.4.2
fsspec	2023.4.0	markupsafe	2.1.1	qt-main	5.15.2	yaml	0.2.5
gast	0.4.0	matplotlib	3.7.1	qt-webengine	5.15.9	yarl	1.8.1
gensim	4.3.0	matplotlib-base	3.7.1	qtwebkit	5.212	zict	2.2.0
gflags	2.2.2	mkl	2023.1.0	re2	2022.04.01	zlib	1.2.13
giflib	5.2.1	requests	2.31.0	regex	2022.7.9	zstd	1.5.5
glib	2.69.1	glog	0.5.0				

7.3 Text to Features

7.3.1 Statistics

Table 6: Token Statistics

Statistic	Value
Total words:	1057679905
Unique tokens:	7554704
Mean count:	140.0
StDev of count:	26028.9
Min count:	1
Q25% count:	1
Median count:	1
Q75% count:	2
Max count:	54996963

Table 7: Token Statistics: Stop Words Excluded

Statistic	Value
Total words:	819282440
Unique tokens:	7551697
Mean count:	108.49
StDev of count:	22564.09
Min count:	1
Q25% count:	1
Median count:	1
Q75% count:	2
Max count:	54996963

Table 8: Token Statistics: Stop Words Excluded and Rare Words Mapped, also Unigram Statistics

Statistic	Value
Total words:	819282440
Unique tokens:	614038
Mean count:	1334.25
StDev of count:	80451.07
Min count:	11
Q25% count:	17
Median count:	35
Q75% count:	110
Max count:	54996963

Table 9: Bigram Statistics

Statistic	Value
Total words:	815770115
Unique tokens:	32074403
Mean count:	25.43
StDev of count:	1981.82
Min count:	1
Q25% count:	1
Median count:	1
Q75% count:	3
Max count:	7107613

Table 10: Bleach Text V Statistics

Statistic	Value
Total words:	819282440
Unique tokens:	35732
Mean count:	22928.54
StDev of count:	719400.06
Min count:	11
Q25% count:	21
Median count:	56
Q75% count:	268
Max count:	79407990

Table 11: Bleach Text L Statistics

Statistic	Value
Total words:	819282440
Unique tokens:	151
Mean count:	5425711.52
StDev of count:	20821079.56
Min count:	11
Q25% count:	42
Median count:	214
Q75% count:	3819
Max count:	151533844

Table 12: Bleach Text S Statistics

Statistic	Value
Total words:	819282440
Unique tokens:	2397
Mean count:	341794.93
StDev of count:	11551470.21
Min count:	11
Q25% count:	19
Median count:	49
Q75% count:	198
Max count:	552500011

Table 13: Bleach Text A Statistics

Statistic	Value
Total words:	819282440
Unique tokens:	8
Mean count:	102410305.0
StDev of count:	229732880.59
Min count:	69
Q25% count:	15361
Median count:	4252620
Q75% count:	32149056.25
Max count:	705001066

Table 14: Bleach Text C Statistics

Statistic	Value
Total words:	819282440
Unique tokens:	22
Mean count:	37240110.91
StDev of count:	146555113.88
Min count:	28
Q25% count:	82
Median count:	5095
Q75% count:	1271121.75
Max count:	705001066

Table 15: Bleach Text ALL Statistics

Statistic	Value
Total words:	819282440
Unique tokens:	36885
Mean count:	22211.81
StDev of count:	650808.64
Min count:	11
Q25% count:	21
Median count:	56
Q75% count:	267
Max count:	79407990

7.3.2 Custom Stopwords

"x", "a1", "r", "onze", "g", "indien", "ten", "wij", "chr", "w", "enof", "welke", "f", "p", "enige", "tenzij", "zoals", "s1", "s3", "eventuele", "volgens", "slechts", "algemene", "huidige", "p3", "verzoek", "s4", "k2", "p1", "p2", "k1", "tussen", "enorm", "betreffende", "gevolg", "uitdrukkelijk", "co", "uitsluitend", "zullen", "aa", "gelieve", "wel", "zie", "eveneens", "eventueel", "geacht", "a2", "graag", "ieder", "rgr", "s2", "elke", "wi", "zover", "fsc", "waarop", "waarvan", "alleen", "xa9n" "vd", "verzoeken", "elk", "daarvan", "alsmede", "jegens", "svp", "m3", "mede", "eo", "inzake" "zb" , "rgr", 'pt', "a2", "dc", "aub", "alsjeblieft"

7.3.3 Overview

Table 16: Overview of the Generated Feature Sets

Feature Set Basen On	BoW Representation	Accumulated Word Vectors	Document Vectors	Bleaching	SVD Applied	Stop Word Removal
Unigram	+				+	+
Bigram	+				+	+
Word2Vec		+				+
Doc2Vec			+			+
Fasttext		+				+
Bert			+			+
Bleach Text L	+			+		+
Bleach Text C	+			+		+
Bleach Text A	+			+		+
Bleach Text S	+			+		+
Bleach Text V	+			+	+	+
Bleach Text ALL	+			+	+	+

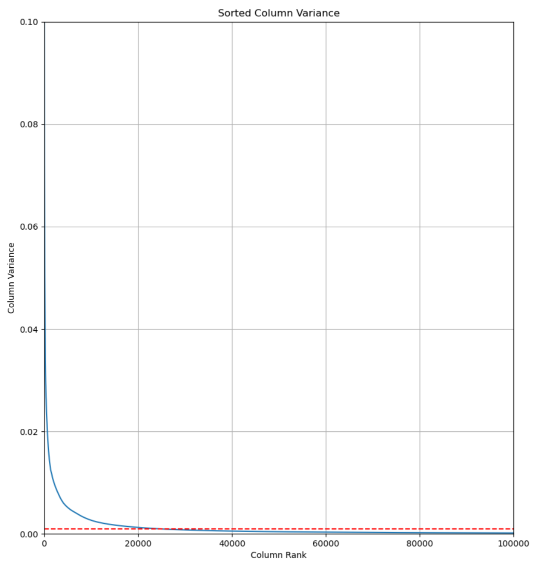
7.3.4 Regular Expressions

Regular expressions define a search pattern. Regular expressions provide a way to match, search, and manipulate sequences of characters in a text. The matched text is replaced by a pattern specific token such as ADD for addresses. Table 17 shows all impletemented replacements.

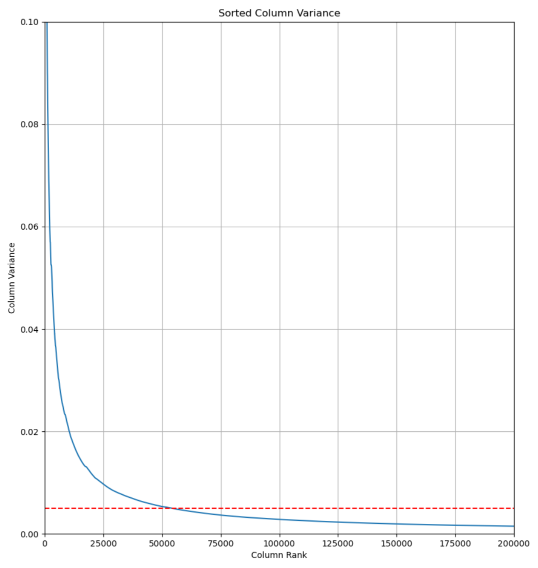
Table 17: Implemented Regular Expressions

Pattern for	Replacement Token	Regular Expression Pattern
URL	URL	https?:\/\/?[a-z0-9]+\.[a-z0-9\/-]+ and www\.[a-z0-9-]+(\.[a-z]{2,3})+
E-mail address	EMA	[a-z0-9\.]+[@][a-z0-9\.-]+\.[a-z]{2,3}
Time	TIM	\d\d?:\d\d?
5+ digit number	NMB	\s?\w*\d{5,}\w*\s?
9+ number sequence	NSQ	(\d\s*){9,}
Symbols	SYM	[a-z0-9(\.]+\\xc\d\\[a-z0-9/\\)\.]+
Adress	ADD	[a-z]*(straat|(...)|street)[a-z]*

7.3.5 Sorted Variances Unigram and Bigram



(a) Sorted Variances Unigrams



(b) Sorted Variances Bigrams

Figure 8: Sorted Variances Unigram and Bigram

7.3.6 Bleaching Text Variations

This section provides an overview of the different bleaching text variants:

- *Bleach Text L*: The abstract tokens in this feature set are created by counting the characters of each token, prefixed by 0.
- *Bleach Text C*: The abstract tokens in this feature set are created by merging all following alphanumeric characters to a single *W*. Every other character remains as it is.
- *Bleach Text A*: The abstract tokens in this feature set are similar to *Bleach Text C*, nevertheless, punctuation is converted to *P*
- *Bleach Text S*: In *Bleach Text S* uppercase characters are transformed to *U*, lowercase characters to *L*, numerical characters to *D* and every other character to *X* to create the abstract tokens in this feature set.
- *Bleach Text V*: In this representation, in order to create the abstract tokens, characters equal to *a, e, i, o* or *u* are converted to *V*. Consonants are converted to *C* and everything else to *O*.

7.4 Predicting Payment Term

7.4.1 Kernel functions

Linear: $K(w, b) = w^T x + b$

Polynomial: $K(w, x) = (\gamma w^T x + b)^N$

Gaussian RBF: $K(w, x) = exp(-\gamma ||x_i - x_j||^n)$

Sigmoid: $K(x_i, x_j) = tanh(\alpha x_i^T x_j + b)$

7.4.2 Overview Classification Algorithms

Table 18: Overview of the used Classification Algorithms

Classifier(s)	Supervised	Self-training	Co-training
K-NN	+		
MLR	+	+	+
SVC	+		+

7.4.3 Test F$_1$-Scores Supervised Learning

Table 19: F$_1$-Scores Supervised Learning

Dataset Based On	(Optional) Hyperparameters	KNN	MLR	SVC
Bert		0.8074	0.7253	0.8211
Bigram		0.8374	0.8930	0.8821
Bleach Text A		0.5742	0.4022	0.4945
Bleach Text ALL		0.8229	0.8015	0.8409
Bleach Text C		0.5909	0.4022	0.4878
Bleach Text L		0.7456	0.4568	0.6346
Bleach Text S		0.7102	0.4767	0.6139
Bleach Text V		0.8223	0.7991	0.8392
Doc2Vec	Length: 1000 & Window: 15	0.4039	0.3947	0.4447
Doc2Vec	Length: 1000 & Window: 5	0.4209	0.4476	0.4727
Doc2Vec	Length: 300 & Window: 15	0.4108	0.4045	0.4451
Doc2Vec	Length: 300 & Window: 5	0.4400	0.4306	0.4994
Fasttext	Length: 100 & Window: 15	0.8230	0.6850	0.7785
Fasttext	Length: 100 & Window: 5	0.8152	0.6828	0.7896
Fasttext	Length: 300 & Window: 15	0.8138	0.7395	0.8158
Fasttext	Length: 300 & Window: 5	0.8103	0.7309	0.8114
Word2Vec	Length: 100 & Window: 15	0.8222	0.6917	0.8065
Word2Vec	Length: 100 & Window: 5	0.8170	0.6778	0.8025
Word2Vec	Length: 300 & Window: 15	0.8182	0.7343	0.8278
Word2Vec	Length: 300 & Window: 5	0.8144	0.7282	0.8249
Unigram		0.8181	0.8676	0.8557

* Did not converge

7.4.4 All Test Results SVC Supervised Learning

Table 20: Test Results SVC

Dataset Based On	(Optional) Hyperparameters	Kernel	C	Accuracy	F1 Score	Cohen's Kappa	F1 Score no other
Bert		rbf	10	0.8288	0.8211	0.7192	0.7569
Bigram		rbf	10	0.8885	0.8821	0.8157	0.8532
Bleach Text A		rbf	10	0.5904	0.4945	0.1540	0.2027
Bleach Text ALL		rbf	10	0.8478	0.8409	0.7495	0.7877
Bleach Text V		rbf	10	0.8462	0.8392	0.7474	0.7879
Bleach Text C		rbf	10	0.5916	0.4878	0.1365	0.1815
Bleach Text L		rbf	10	0.6779	0.6346	0.4064	0.4773
Bleach Text S		rbf	10	0.6646	0.6139	0.3742	0.4304
Doc2Vec	Length: 1000 & Window: 15	rbf	10	0.5539	0.4447	0.0531	0.1207
Doc2Vec	Length: 1000 & Window: 5	rbf	10	0.5605	0.4727	0.1071	0.1839
Doc2Vec	Length: 300 & Window: 15	rbf	10	0.5518	0.4451	0.0580	0.1178
Doc2Vec	Length: 300 & Window: 5	rbf	10	0.5767	0.4994	0.1607	0.2402
Fasttext	Length: 100 & Window: 15	rbf	10	0.7939	0.7785	0.6543	0.6743
Fasttext	Length: 100 & Window: 5	rbf	10	0.8022	0.7896	0.6704	0.6897
Fasttext	Length: 300 & Window: 15	rbf	10	0.8238	0.8158	0.7099	0.7363
Fasttext	Length: 300 & Window: 5	rbf	10	0.8197	0.8114	0.7044	0.7360
Word2Vec	Length: 100 & Window: 15	rbf	10	0.8155	0.8065	0.6953	0.7275
Word2Vec	Length: 100 & Window: 5	rbf	10	0.8122	0.8025	0.6894	0.7252
Word2Vec	Length: 300 & Window: 15	rbf	10	0.8333	0.8278	0.7305	0.7719
Word2Vec	Length: 300 & Window: 5	rbf	10	0.8313	0.8249	0.7252	0.7617
Unigram		rbf	10	0.8661	0.8557	0.7765	0.8027

7.4.5 All Test Results MLR Supervised Learning

Table 21: Test Results MLR

Dataset Based On	(Optional) Hyperparameters	Algorithm	C	Accuracy	F1 Score	Cohen's Kappa	F1 Score no other
Bert		sag	10	0.7504	0.7253	0.5685	0.6016
Bigram		sag	10	0.8980	0.8930	0.8335	0.8698
Bleach Text A		newton-cg	0.1	0.5601	0.4022	0	0
Bleach Text ALL		sag	10	0.8060	0.8015	0.6876	0.7351
Bleach Text C		newton-cg	0.1	0.5601	0.4022	0	0
Bleach Text L		newton-cg	10	0.5680	0.4568	0.0815	0.1316
Bleach Text S		newton-cg	10	0.5821	0.4767	0.1148	0.1693
Bleach Text V		newton-cg	10	0.8047	0.7991	0.6844	0.7381
Doc2Vec	Length 1000 & Window 15	newton-cg	1	0.4092	0.3947	0.0202	0.2144
Doc2Vec	Length 1000 & Window 5	lbfgs	0.1	0.5323	0.4476	0.0581	0.1681
Doc2Vec	Length 300 & Window 15	newton-cg	1	0.4685	0.4045	0.0095	0.1113
Doc2Vec	Length 300 & Window 5	newton-cg	0.1	0.5406	0.4306	0.0304	0.0936
Fasttext	Length 100 & Window 15	newton-cg	10	0.7094	0.6850	0.4976	0.5555
Fasttext	Length 100 & Window 5	lbfgs	10	0.7106	0.6828	0.4953	0.5491
Fasttext	Length 300 & Window 15	lbfgs	10	0.7521	0.7395	0.5858	0.6421
Fasttext	Length 300 & Window 5	sag	10	0.7454	0.7309	0.5743	0.6303
Word2Vec	Length 100 & Window 15	sag	10	0.7156	0.6917	0.5077	0.5499
Word2Vec	Length 100 & Window 5	newton-cg	10	0.7040	0.6778	0.4840	0.5487
Word2Vec	Length 300 & Window 15	newton-cg	10	0.7475	0.7343	0.5777	0.6358
Word2Vec	Length 300 & Window 5	sag	10	0.7438	0.7282	0.5688	0.6320
Unigram		sag	10	0.8735	0.8676	0.7947	0.8289

* Did not converge

7.4.6 All Test Results KNN Supervised Learning

Table 22: Test Results KNN

Dataset Based On	(Optional) Hyperparameters	Neighbors	Weight	Distance Measure	Accuracy	F1 Score	Cohen's Kappa	F1 Score no other
Bert		3	distance	cosine	0.8072	0.8074	0.6964	0.7659
Bigram		3	distance	cosine	0.8391	0.8374	0.7435	0.7992
Bleach Text A		33	distance	euclidean	0.6144	0.5742	0.3116	0.3793
Bleach Text ALL		3	distance	cosine	0.8250	0.8229	0.7819	0.7175
Bleach Text C		33	distance	manhattan	0.6298	0.5909	0.3362	0.4020
Bleach Text L		3	distance	manhattan	0.7471	0.7456	0.5970	0.6849
Bleach Text S		3	distance	manhattan	0.7143	0.7102	0.5370	0.6050
Bleach Text V		3	distance	cosine	0.8242	0.8223	0.7863	0.7161
Doc2Vec	Length: 1000 & Window: 15	3	distance	euclidean	0.4851	0.4039	-0.0136	0.0981
Doc2Vec	Length: 1000 & Window: 5	3	distance	euclidean	0.4585	0.4209	0.0355	0.1945
Doc2Vec	Length: 300 & Window: 15	3	uniform	cosine	0.5232	0.4108	-0.0055	0.0646
Doc2Vec	Length: 300 & Window: 5	3	distance	euclidean	0.4689	0.4400	0.0731	0.2471
Fasttext	Length: 100 & Window: 15	3	distance	cosine	0.8242	0.8230	0.7209	0.7813
Fasttext	Length: 100 & Window: 5	3	distance	cosine	0.8151	0.8152	0.7078	0.7605
Fasttext	Length: 300 & Window: 15	3	distance	cosine	0.8138	0.8138	0.7052	0.7728
Fasttext	Length: 300 & Window: 5	3	distance	cosine	0.8105	0.8103	0.7000	0.7640
Word2Vec	Length: 100 & Window: 15	3	distance	cosine	0.8226	0.8222	0.7188	0.7779
Word2Vec	Length: 100 & Window: 5	3	distance	cosine	0.8172	0.8170	0.7095	0.7677
Word2Vec	Length: 300 & Window: 15	3	distance	cosine	0.8180	0.8182	0.7132	0.7887
Word2Vec	Length: 300 & Window: 5	3	distance	cosine	0.8151	0.8144	0.7065	0.7676
Unigram		3	distance	cosine	0.8209	0.8181	0.7760	0.7118

7.4.7 Differences Between F_1-Score and F_1-Score Without the "Other" Class

Table 23: F_1-Scores KNN

Dataset Based On	(Optional) Hyperparameters	Neighbors	Weight	Distance Measure	F1 Score	F1 Score no other	Absolute Difference	Difference As Percentage
Bigram		3	distance	cosine	0.84	0.80	0.04	4.55
Fasttext	Length: 100 & Window: 15	3	distance	cosine	0.82	0.78	0.04	5.06
Bleach Text ALL		3	distance	cosine	0.82	0.78	0.04	4.98
Bleach Text V		3	distance	cosine	0.82	0.72	0.11	12.91
Word2Vec	Length: 100 & Window: 15	3	distance	cosine	0.82	0.78	0.04	5.39
Word2Vec	Length: 300 & Window: 15	3	distance	cosine	0.82	0.79	0.03	3.60
Unigram		3	distance	cosine	0.82	0.71	0.11	12.99
Word2Vec	Length: 100 & Window: 5	3	distance	cosine	0.82	0.77	0.05	6.03
Fasttext	Length: 100 & Window: 5	3	distance	cosine	0.82	0.76	0.05	6.71
Word2Vec	Length: 300 & Window: 5	3	distance	cosine	0.81	0.77	0.05	5.75
Fasttext	Length: 300 & Window: 15	3	distance	cosine	0.81	0.77	0.04	5.03
Fasttext	Length: 300 & Window: 5	3	distance	cosine	0.81	0.76	0.05	5.71
Bert		3	distance	cosine	0.81	0.77	0.04	5.14
Bleach Text L		3	distance	manhattan	0.75	0.68	0.06	8.14
Bleach Text S		3	distance	manhattan	0.71	0.61	0.11	14.80
Bleach Text C		33	distance	manhattan	0.59	0.40	0.19	31.96
Bleach Text A		33	distance	euclidean	0.57	0.38	0.19	33.94
Doc2Vec	Length: 300 & Window: 5	3	distance	euclidean	0.44	0.25	0.19	43.83
Doc2Vec	Length: 1000 & Window: 5	3	distance	euclidean	0.42	0.19	0.23	53.78
Doc2Vec	Length: 300 & Window: 15	3	uniform	cosine	0.41	0.06	0.35	84.27
Doc2Vec	Length: 1000 & Window: 15	3	distance	euclidean	0.40	0.10	0.31	75.71

Table 24: F_1-Scores SVC

Dataset Based On	(Optional) Hyperparameters	Kernel	C	F1 Score	F1 Score no other	Absolute Difference	Difference As Percentage
Bigram		rbf	10	0.88	0.85	0.03	3.28
Unigram		rbf	10	0.86	0.80	0.05	6.19
Bleach Text ALL		rbf	10	0.84	0.79	0.05	6.33
Bleach Text V		rbf	10	0.84	0.79	0.05	6.12
Word2Vec	Length: 300 & Window: 15	rbf	10	0.83	0.77	0.06	6.76
Word2Vec	Length: 300 & Window: 5	rbf	10	0.82	0.76	0.06	7.66
Bert		rbf	10	0.82	0.76	0.06	7.82
Fasttext	Length: 300 & Window: 15	rbf	10	0.82	0.74	0.08	9.76
Fasttext	Length: 300 & Window: 5	rbf	10	0.81	0.74	0.08	9.29
Word2Vec	Length: 100 & Window: 15	rbf	10	0.81	0.73	0.08	9.80
Word2Vec	Length: 100 & Window: 5	rbf	10	0.80	0.73	0.08	9.63
Fasttext	Length: 100 & Window: 5	rbf	10	0.79	0.69	0.10	12.65
Fasttext	Length: 100 & Window: 15	rbf	10	0.78	0.67	0.10	13.38
Bleach Text L		rbf	10	0.63	0.48	0.16	24.79
Bleach Text S		rbf	10	0.61	0.43	0.18	29.90
Doc2Vec	Length: 300 & Window: 5	rbf	10	0.50	0.24	0.26	51.90
Bleach Text A		rbf	10	0.49	0.20	0.29	59.00
Bleach Text C		rbf	10	0.49	0.18	0.31	62.79
Doc2Vec	Length: 1000 & Window: 5	rbf	10	0.47	0.18	0.29	61.10
Doc2Vec	Length: 300 & Window: 15	rbf	10	0.45	0.12	0.33	73.53
Doc2Vec	Length: 1000 & Window: 15	rbf	10	0.44	0.12	0.32	72.87

Table 25: F_1-Scores MLR

Dataset Based On	(Optional) Hyperparameters	Algorithm	C	F1 Score	F1 Score no other	Absolute Difference	Difference As Percentage
Bigram		sag	10	0.89	0.87	0.02	2.60
Unigram		sag	10	0.87	0.83	0.04	4.46
Bleach Text ALL		sag	10	0.80	0.74	0.07	8.29
Bleach Text V		newton-cg	10	0.80	0.74	0.06	7.63
Fasttext	Length 300 & Window 15	lbfgs	10	0.74	0.64	0.10	13.17
Word2Vec	Length 300 & Window 15	newton-cg	10	0.73	0.64	0.10	13.42
Fasttext	Length 300 & Window 5	sag	10	0.73	0.63	0.10	13.76
Word2Vec	Length 300 & Window 5	sag	10	0.73	0.63	0.10	13.20
Bert		sag	10	0.73	0.60	0.12	17.06
Word2Vec	Length 100 & Window 15	sag	10	0.69	0.55	0.14	20.50
Fasttext	Length 100 & Window 15	newton-cg	10	0.69	0.56	0.13	18.91
Fasttext	Length 100 & Window 5	lbfgs	10	0.68	0.55	0.13	19.57
Word2Vec	Length 100 & Window 5	newton-cg	10	0.68	0.55	0.13	19.05
Bleach Text S		newton-cg	10	0.48	0.17	0.31	64.49
Bleach Text L		newton-cg	10	0.46	0.13	0.33	71.18
Doc2Vec	Length 1000 & Window 5	lbfgs	0.1	0.45	0.17	0.28	62.45
Doc2Vec	Length 300 & Window 5	newton-cg	0.1	0.43	0.09	0.34	78.25
Doc2Vec	Length 300 & Window 15	newton-cg	1	0.40	0.11	0.29	72.49
Bleach Text C		newton-cg	0.1	0.40	0.00	0.40	100.00
Bleach Text A		newton-cg	0.1	0.40	0.00	0.40	100.00
Doc2Vec	Length 1000 & Window 15	newton-cg	1	0.39	0.21	0.18	45.69

* Did not converge

7.4.8 All Test Results Self-Training

Table 26: Test Results Self-Training using MLR

Dataset Based On	Iteration	Threshold	algorithm	C	Accuracy	F1 Score	Cohen's Kappa	F1 Score no other
Bigram	1	0.999	sag	10	0.8980	0.8930	0.8335	0.8698
Bigram	2	0.999	sag	10	0.8852	0.8758	0.8083	0.8371
Bigram	1	0.999999	sag	10	0.8980	0.8930	0.8335	0.8698
Bigram	2	0.999999	sag	10	0.8897	0.8818	0.8166	0.8477
Bigram	3	0.999999	sag	10	0.8864	0.8773	0.8104	0.8392
Bigram	1	0.999999999	sag	10	0.8980	0.8930	0.8335	0.8698
Bigram	2	0.999999999	sag	10	0.8918	0.8846	0.8208	0.8537
Bigram	3	0.999999999	sag	10	0.8893	0.8811	0.8158	0.8463
Bigram	4	0.999999999	sag	10	0.8864	0.8777	0.8101	0.8400
Bigram	5	0.999999999	sag	10	0.8852	0.8754	0.8079	0.8380

7.4.9 All Iteration Results Self-Training

Table 27: Iteration Results Self-Training using MLR

Dataset Based On	Iteration	Threshold	algorithm	C	Accuracy	F1 Score	Cohen's Kappa
Bigram	2	0.999	sag	10	0.9943	0.9940	0.9923
Bigram	2	0.999999	sag	10	0.9925	0.9921	0.9901
Bigram	3	0.999999	sag	10	0.9942	0.9938	0.9921
Bigram	2	0.999999999	sag	10	0.9875	0.9867	0.9836
Bigram	3	0.999999999	sag	10	0.9930	0.9929	0.9905
Bigram	4	0.999999999	sag	10	0.9940	0.9936	0.9913
Bigram	5	0.999999999	sag	10	0.9941	0.9937	0.9915

7.4.10 All Test Results Co-Training

Table 28: Test Results Co-Training MLR

Dataset Based on	Iteration	Threshold	Algorithm	C	Accuracy	F1 Score	Cohen's Kappa	F1 Score no other
Bigram	1	125000	sag	10	0.8980	0.8930	0.8335	0.8698
Bigram	2	125000	sag	10	0.8852	0.8775	0.8098	0.8504
Bigram	1	50000	sag	10	0.8980	0.8930	0.8335	0.8698
Bigram	2	50000	sag	10	0.8947	0.8886	0.8270	0.8649
Bigram	3	50000	sag	10	0.8889	0.8819	0.8164	0.8519
Bigram	4	50000	sag	10	0.8897	0.8822	0.8174	0.8504
Bigram	1	25000	sag	10	0.8980	0.8930	0.8335	0.8698
Bigram	2	25000	sag	10	0.8959	0.8901	0.8293	0.8677
Bigram	3	25000	sag	10	0.8959	0.8900	0.8292	0.8670
Bigram	4	25000	sag	10	0.8914	0.8845	0.8207	0.8568
Bigram	5	25000	sag	10	0.8914	0.8845	0.8210	0.8551
Bigram	6	25000	sag	10	0.8918	0.8845	0.8211	0.8552

Table 29: Test Results Co-Training SVC

Dataset Based On	Iteration	Threshold	Kernel	C	Accuracy	F1 Score	Cohen's Kappa	F1 Score no other
Bigram	1	125000	rbf	10	0.8885	0.8821	0.8157	0.8532
Bigram	2	125000	rbf	10	0.8934	0.8866	0.8236	0.8577
Bigram	1	50000	rbf	10	0.8885	0.8821	0.8157	0.8532
Bigram	2	50000	rbf	10	0.8889	0.8813	0.8152	0.8495
Bigram	3	50000	rbf	10	0.8934	0.8866	0.8235	0.8577
Bigram	4	50000	rbf	10	0.8930	0.8862	0.8231	0.8569
Bigram	1	25000	rbf	10	0.8885	0.8821	0.8157	0.8532
Bigram	2	25000	rbf	10	0.8905	0.8832	0.8181	0.8526
Bigram	3	25000	rbf	10	0.8901	0.8827	0.8174	0.8526
Bigram	4	25000	rbf	10	0.8922	0.8853	0.8214	0.8555
Bigram	5	25000	rbf	10	0.8934	0.8862	0.8229	0.8572
Bigram	6	25000	rbf	10	0.8943	0.8875	0.8250	0.8587

7.4.11 All Iteration Results Co-Training

Table 30: Iteration Results Co-Training MLR

Dataset Based On	Iteration	Threshold	Algorithm	C	Accuracy	F1 Score	Cohen's Kappa
Bigram	2	125000	sag	10	0.9903	0.9896	0.9805
Bigram	2	50000	sag	10	0.9778	0.9758	0.9600
Bigram	3	50000	sag	10	0.9871	0.9861	0.9749
Bigram	4	50000	sag	10	0.9905	0.9899	0.9835
Bigram	2	25000	sag	10	0.9637	0.9605	0.9367
Bigram	3	25000	sag	10	0.9776	0.9758	0.9564
Bigram	4	25000	sag	10	0.9837	0.9826	0.9619
Bigram	5	25000	sag	10	0.9855	0.9843	0.9631
Bigram	6	25000	sag	10	0.9872	0.9861	0.9653