

Teemu Eerola

ENHANCING AUTOMATIC INVOICE CODING PERFORMANCE WITH UNSTRUCTURED DATA

A feature extraction approach

Master of Science Thesis
Faculty of Information Technology and Communications
Examiners: D.Sc. (Tech.) Antti Stenvall
Prof. Jyrki Nummenmaa
March 2024

ABSTRACT

Teemu Eerola: Enhancing Automatic Invoice Coding Performance with Unstructured Data
Master of Science Thesis
Tampere University
Master's Programme in Information Technology
March 2024

As a part of invoice processing, companies are documenting their spend to cope with financial audits, tax regulations, and their spend management purposes in a process called invoice coding. This process involves selecting coding values from a predefined list for each coding dimension describing the different aspects of spend. However, the manual coding process can be time-consuming and the traditional invoice automation processes still fail to automate coding completely.

The invoice coding problem can be formulated as a supervised classification task and a machine learning model can be trained on historical invoices to predict coding values for new invoices. This study explores how to incorporate the textual data of the invoice to improve the coding predictions. This data has not been previously used in the baseline system, because the lack of standardization in invoices makes it difficult to capture. This work studies the feasible feature extraction methods and assesses the performance against the baseline result that uses only invoice header data.

Experiments are conducted across datasets from seven companies with varying invoice origins, processes, and languages. Different coding dimensions bring diverse aspects of coding to experiments like capturing related words or semantic meaning. The results showed that incorporating textual data can improve accuracy by up to 21 percentage points over the baseline. The most robust performance improvement was obtained by latent semantic analysis features.

This thesis provides a practical pathway for implementing these enhancements in production and lays the groundwork for future advancements leveraging the textual data of invoice content.

Keywords: Invoice automation, Accounts payable, Invoice coding, Machine learning, XGBoost, Latent semantic analysis, Word2Vec, FastText

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Teemu Eerola: Automaattisten tiliöintiennusteiden parantaminen hyödyntämällä epämuodollista tietoa

Diplomityö

Tampereen yliopisto

Tietotekniikan DI-ohjelma

Maaliskuu 2024

Osana laskujen käsittelyä yritykset dokumentoivat kulujaan kirjanpitoa varten sekä seuratakseen kuluja. Tätä prosessissa kutsutaan tiliöinniksi. Yleisimmin tiliöintiin kuuluu tiliöintiarvojen valitseminen ennalta määritetystä luettelosta jokaiselle tiliöintidimensiolle, jotka kuvaavat kulujen eri osa-alueita. Manuaalinen tiliöinti on aikaa vievää ja perinteiset laskuautomaatioprosessit eivät onnistu täysin automatisoimaan tiliöintiä.

Laskun tiliöinti voidaan muotoilla ohjatuksi luokittelutehtäväksi, jossa koneoppimismalli voidaan kouluttaa historiallisilla laskuilla ennustamaan tiliöintiä uusille laskuille. Tässä tutkimuksessa selvitetään, miten laskulla olevaa tekstiä voidaan hyödyntää tiliöintiennusteen parantamiseksi. Tätä tietoa ei ole aiemmin käytetty tiliöinnin ennusteissa, koska laskujen standardoinnin puute vaikeuttaa tietojen tallentamista relaationa. Tässä työssä tutkitaan käyttökelpoisia ominaisuuksien erottelumenetelmiä ja luokittelumallin suorituskyvyn parannusta lähtötasoon, jossa käytetään vain laskun otsikkotietoja.

Kokeita suoritetaan seitsemän eri yrityksen aineistoista, joiden laskujen alkuperä, prosessit ja käytetyt kielet vaihtelevat. Eri tiliöintidimensiot tuovat kokeiluihin erilaisia tiliöinnin osa-alueita kuten tiettyjen tiliöintiin liittyvien sanojen löytämistä tai semanttisten merkitysten hyödyntämistä. Tulokset osoittivat, että laskun tekstitiedon käyttäminen voi parantaa tiliöintiennusteiden tarkkuutta jopa 21 prosenttiyksikköä lähtötasoon nähden. Parhaat ja vakaimmat suorituskyvyn parannukset saadaan käyttäen ominaisuuksia erotteluun *latent semantic analysis* -menetelmää.

Tämä opinnäytetyö esittelee käytännöllisen menetelmän tiliöintiennusteen parantamiseen, joka on mahdollista toteuttaa tuotantoympäristössä. Työ myös luo pohjaa tuleville edistysaskeleille laskun tekstitiedon hyödyntämisessä erilaisissa laskuautomaation tehtävissä.

Avainsanat: Laskuautomaatio, Tiliöinti, Koneoppiminen

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

PREFACE

This thesis project was done for Basware as part of a broader goal of the company to utilize data and AI. I want to thank every person from Basware side for making this project possible. It has been a wonderful opportunity for me to gain a deeper understanding of invoice automation and great spot to showcase the ML skills I have learned throughout my studies. I hope this thesis also help the stakeholders in Basware to identify the potential future ML applications and requirements for those.

Special mentions belong to Antti Stenvall, supervisor of the thesis, for insightful discussions, comments, \LaTeX -mentorship, and pointing out even the tiniest details, Olli-Pekka for organizing the opportunity for this project, Anna for initial research idea, setting the expectations, and interest towards the project even after leaving Basware, as well as Anders and Vien for sharing your knowledge. For directing the academic side of the work, I want to thank professor Jyrki Nummenmaa.

Erityiskiitos kuuluu myös Camillalle, joka on koko projektin läpi sinnikkäästi muistuttanut päästämään ajatukset välillä myös muualle kuin laskujen tiliöinnin ja koneoppimismenetelmien yksityiskohtiin.

*"Data matures like wine, applications like fish."
- James Governor*

Tampere, 16th March 2024

Teemu Eerola

CONTENTS

1.	Introduction	1
1.1	Objectives, scope, and structure	2
2.	Background	5
2.1	Invoice process and coding	5
2.1.1	Coding in accounting	7
2.1.2	Determining the correct coding	9
2.1.3	AP automation	9
2.2	Receiving invoices	11
2.3	Related works	13
2.4	SmartCoding - Basware's solution to coding suggestions	14
3.	Methods	16
3.1	Machine learning approach	16
3.2	Feature extraction from invoice content	18
3.2.1	Bag of Words	20
3.2.2	Latent semantic analysis	21
3.2.3	Word2Vec	23
3.2.4	FastText.	26
3.3	Classification model.	28
3.3.1	Multi-class predictions with XGBoost	28
3.3.2	Finding the optimal tree structure.	29
3.3.3	Contribution of XGBoost	32
3.4	Evaluation methods	32
3.4.1	Evaluation metrics	33
3.4.2	Stratified cross validation	35
4.	Implementation and Results	37
4.1	Dataset Description	37
4.2	Preprocessing	40
4.2.1	Invoice header preprocessing	41
4.2.2	Preprocessing of invoice content text	41
4.3	Implementation details.	42
4.4	Comparison of feature extraction techniques	44
4.5	Performance gains from using invoice content data	47
4.5.1	Results for account code hierarchy	49
4.6	Including non-E-invoices to data	50

4.7 Methodology critique	53
5. Conclusions	55
References	57
Appendix A: Invoice XML	60

GLOSSARY

AP	Accounts payable
BERT	Bidirectional encoder representations from transformers
BoW	Bag of Words
IFRS	International financial reporting standards
LLM	Large language model
LSA	Latent semantic analysis
ML	Machine learning
NLP	Natural language processing
OCR	Optical character recognition
P2P	Purchase to pay
PCA	Principal component analysis
PO	Purchase order
SVD	Singular value decomposition
SVM	Support vector machine
TF-IDF	Term frequency inverse document frequency

1. INTRODUCTION

The processing of invoices, like many other software applications, has shifted to cloud-based services that users access through their web browsers. Software operating in the cloud provides a solid foundation for the development of machine learning systems, as it offers access to more information about customer behavior, enables the identification of issues, and provides data and computing capacity for training machine learning models. The invoice automation system developed by Basware, the client of this thesis, is one of the cloud-based solutions available in the global market, primarily targeting businesses dealing with large invoice volumes.

Although invoice automation aims to reduce the human effort required in invoice processing, there are still stages in the process that necessitate human intervention. One of the most manual and labor-intensive tasks is the determination of the invoice's coding line. In accounting, one or more coding lines are assigned to an invoice to divide the total spend of the invoice to coding dimensions.

Each coding line uses a set of coding dimensions and the values of dimensions provide information for the company's accounting and bookkeeping purposes. Examples of coding dimensions include *account code*, which indicates what was purchased on the invoice, and *cost center*, which indicates the entity responsible for the costs on the invoice.

Most coding dimensions have a predefined list of values, from which the appropriate value for the invoice is selected. Selecting the correct value requires an interpretation of the information present on the invoice and an understanding of the company's invoice coding practices. When the invoice data and the final coding lines are stored in the invoice automation software, the problem can be formulated as a multi-class classification task and supervised machine learning methods can be utilized to create a coding prediction system. Although invoice coding cannot be fully automated through machine learning, suggesting likely dimension values significantly increases manual coding efficiency.

This work builds upon a previous machine learning based coding prediction system developed by Basware, expanding it to incorporate additional textual data of the invoice. The textual, previously unutilized data from the invoice content includes information like the names and descriptions of the goods and services and various reference information. That additional data of the invoice content is used by the person who manually assigns

the coding but is currently unutilized by the coding prediction system.

Because invoices are created and transferred in various formats such as e-invoices or PDFs and created by various suppliers, the structure of the invoice and the exact syntax of the references and items vary. Therefore, this work utilizes the textual data of invoices as an unstructured data source and seeks how to use that for improving the coding predictions. Leveraging the invoice content data as unstructured data allows the system to utilize text from invoices regardless of the origin, which broadens the scope of potential customers benefitting from this new feature.

There have been studies on predicting coding values using machine learning [1, 3, 4, 9, 12, 16, 21] , but the datasets and implementations used have not been openly published due to commercial and privacy reasons. Previous research has focused on predicting account codes, while some studies have also addressed the prediction of tax codes as part of their research. The majority of prior research efforts were tailored toward streamlining the invoice processing of small-scale enterprises handling modest invoice volumes. Some previous research has utilized the textual descriptions of invoice line items with promising results.

In contrast to previous research, this work is not limited to only account and tax code predictions but investigates the utility of invoice content data for predictions of each coding dimension used by the customer. The previous research also assumed that structured information from invoice lines is available, whereas this work takes a more general approach and tries to extract information from unstructured invoice data. The dataset used in this work seems to have much more variation in terms of used dimensions, dataset size, and invoice formats than previous works that set unique requirements for the scalability and robustness of the developed solution.

Figure 1.1 gives an example invoice that has the invoice header data highlighted. In contrast to other types of documents like news articles, the header consists of all the information common to the invoice, not only the bolded text at the top of the document. Other text in the invoice than highlighted data is the invoice content data, that is utilized in this work along with the header. In general, an invoice is a document issued by the supplier to the buyer describing the products, quantities, and prices for ordered items or services along with payment information like payment terms. The format of the invoice document can be anything based on the supplier's fondness. This example invoice is carried throughout this work to provide examples of invoice processing.

1.1 Objectives, scope, and structure

The purpose of this thesis is to examine how unstructured textual data from the invoices can be used to improve the machine learning based invoice coding performance. As the

Software Engineering Center Oy Mannerheimintie 123, 00001 Helsinki Phone (123) 456-7890 Fax (123) 456-7891		SEC OY	
INVOICE NO. 1002		DATE 01.02.2023	
BILL TO Basware Oyj BW00 Hämeenkatu 1 Tampere	SHIP TO Same as recipient	CUSTOMER REFERENCE Jane Doe Cost center 6013	TERMS OF PAYMENT Net 7 Overdue rate: 8%
QUANTITY	DESCRIPTION	UNIT PRICE	TOTAL
12	Software expenses towards product development	120.00 €, VAT 0%	1440.00 €, VAT 0%
1	License costs	99.99 €, VAT 0%	99.99 €, VAT 0%
SUBTOTAL			1539.99 €
VAT 24 %			369.60 €
SHIPPING & HANDLING			0.00 €
TOTAL DUE BY DATE 8.2.2023			1909.59 €
Thank you for your business!			

Figure 1.1. Example invoice. Data that is considered to be header data is marked with red.

classification model used can not understand raw text, the text from the invoice needs to be converted to numerical features. This process is typically called feature extraction. The objective of this thesis is to make comparisons of different feature extraction techniques appearing in the literature related to coding predictions. Also, extensions of these feature extraction techniques are proposed. The used methods are compared against each other and to method that utilizes only the invoice header data.

The main research questions to answer in this work are

1. Which feature extraction techniques are most effective in converting textual data from invoices into features suitable for coding predictions?
2. How does adding features from the invoice content affect the performance of the coding predictions compared to using the header data only?
3. Do some coding dimensions benefit more from utilizing the invoice content features?

With these research questions in mind, datasets from 7 different companies having vary-

ing sizes, invoice origins, and coding dimensions are extracted. Experiments are conducted utilizing various feature extraction techniques and results are evaluated by comparing the classification performance.

This work is limited to taking the invoice header data along with the unstructured content data to make coding predictions with the classification algorithm and finding a proper feature extraction technique for the invoice content. Extracting the text from PDF or paper invoices is a separate challenge not addressed within the scope of this work. Additionally, the intention is solely to expand the existing coding prediction system and therefore, improvements to its implementation are not investigated. This work focuses on bringing the additional features from unstructured invoice data to the classification model and therefore regards the approaches of finding the relevant named entities or coding values.

The thesis is structured as follows: Chapter two provides background information on invoice processing including accounts payable, receiving invoices, and invoice coding. The chapter also introduces the previous research on the topic, as well as Basware's machine learning based coding prediction system. In Chapter three, the overall machine learning pipeline to provide coding suggestions is formulated. As part of the pipeline, the feature extraction methods used in the implementation are discussed on a theoretical level and the necessary metrics for evaluating the techniques are introduced. Chapter four describes the used datasets and implementation of the experimental part of the thesis. The results of the experiments are shown and commented on in the same chapter with a critique of the used techniques and assumptions at the end. Finally, the thesis concludes by summarizing the findings and suggesting topics for future research.

2. BACKGROUND

This chapter takes a more detailed look at invoice processing, invoice automation, and coding as part of it. The origins of invoices and standardization of electronic invoicing are discussed to give insights into the used data sources. In addition related works are introduced including Basware's solution *SmartCoding*, for coding row suggestion with machine learning.

2.1 Invoice process and coding

To explain the invoice process, it is necessary to understand the broad picture of where invoices come from and why invoices are sent. To start with basic trading transactions between supplier and business or company are explained starting from ordering items.

To get needed services, raw materials, or products businesses place orders to suppliers. Seeking suitable suppliers to deliver the key materials or services and negotiating deals is the basis of a successful business. However, all purchases are not critical to business and there is no need for lengthy contract negotiations when casually ordering for example buns and coffee for meetings. For some items like rent or electricity, the deal made with a supplier can be seen as an order, that causes recurring costs.

After the supplier gets an order, it supplies the ordered items or goods to the business. To request payment, the supplier sends an invoice to the business defining the items, quantities, price, tax, payment term, and payment information among other fields required by the local tax authority like invoice date and number [31]. Business gets the items ordered along with the invoice and then takes needed actions to verify the asked amount of payment matches the delivered items and is in line with what was ordered. The business then sends the payment to the supplier with the information specified on the invoice. This basic flow of orders, items, invoices, and payments called purchase to pay (P2P) is presented in Figure 2.1.

The actions of receiving, processing, and paying invoices inside a business is called Accounts Payable (AP) [25, ch. 1]. The manual way of processing one invoice is presented in Figure 2.2. In the figure, a supplier creates an invoice to request payment for delivered goods. Invoice is delivered to business via e-invoice network, by email, or even by fax or

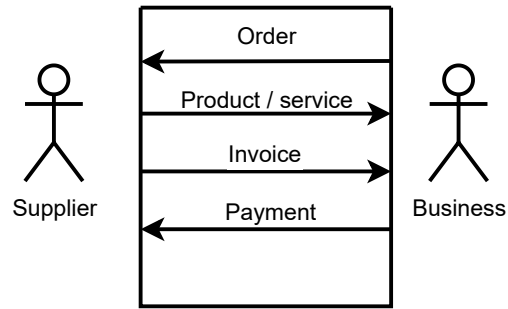


Figure 2.1. The basic flows of orders, items, invoices, and payments between supplier and business in purchase to pay process.

paper. The business then processes the invoice by first checking that the invoice matches the ordered items and that the invoice is valid. Also, quality inspections of items can be made at this point. If the order associated with the invoice is made on ERP or a separate purchase system, it is already coded and reviewed in the purchase requisition phase and no further processing is needed [25, ch. 1] (see *matching* in Section 2.1.3).

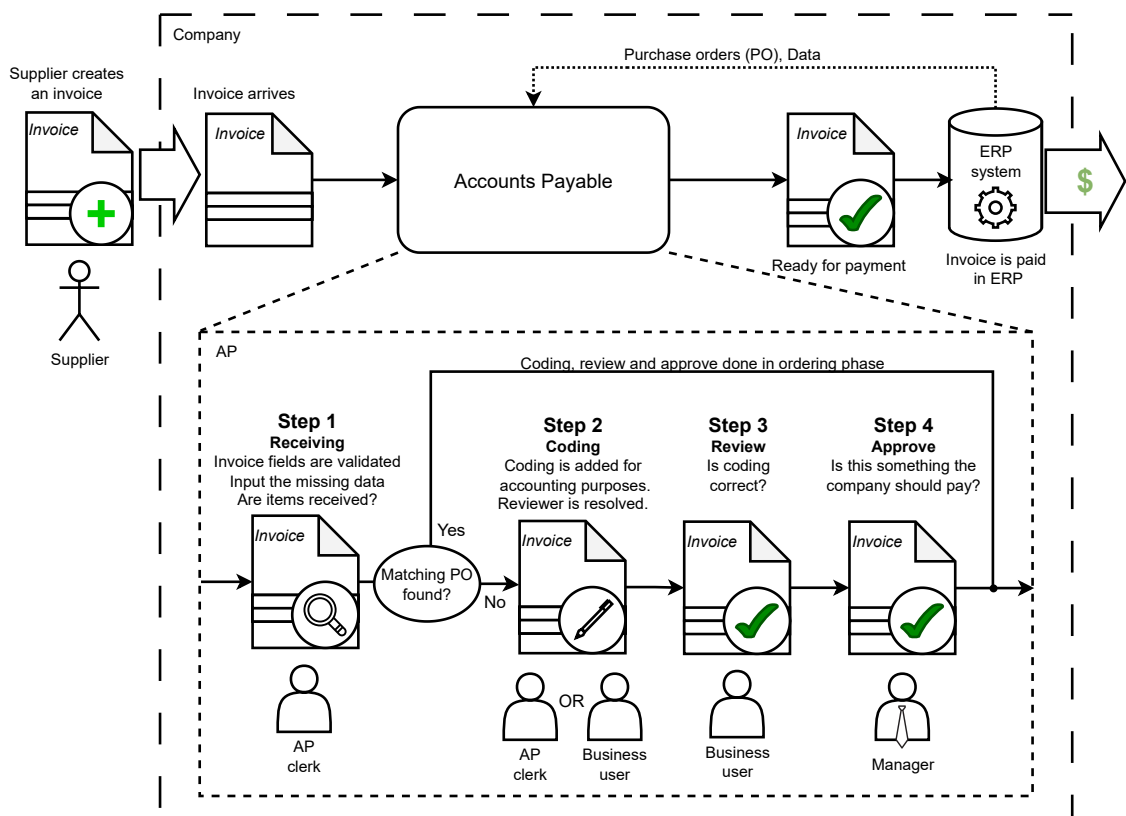


Figure 2.2. The simplified process of a company making orders, receiving, processing, and paying invoices from an AP point of view. Note that many steps can be trivially automated as seen in Section 2.1.3.

For invoices without purchase orders (PO), coding values are added to the invoice after receiving for accounting purposes and the reviewer of the invoice is resolved. Finally, the invoice is approved by the responsible person according to the business rules the

business has set. The business rules might for example consist of the internal approval workflow or limits of accepting sum deviations from the corresponding order to ensure that the spend is authorised.

Receiving of the invoices is done by the AP department by AP clerks. AP clerks are accounting specialists and therefore are more efficient in processing invoices. Coding is usually done by the AP clerk and then reviewed by the person who ordered the item usually noted as a business user. In some cases, the task of coding is done fully by the business user. It is also possible that AP clerks code some of the dimensions and business users the rest.

2.1.1 Coding in accounting

At the highest level, coding is a way for businesses to gather information for accounting. Keeping track of spend is a key factor for a successful business. There are also laws and standards that businesses must follow in their accounting process. To make a distinction, accounting is typically divided into two categories: financial accounting and management accounting [28].

Financial accounting aims to interpret information to external sources such as investors and is subject to regulation and standards. International financial reporting standards (IFRS) [13], requires businesses to present their financial results in a standardized way, and local tax regulations force them to keep track of the amount of tax paid. Management accounting however aims to provide information to internal management to plan and operate the business efficiently and is therefore unstandardized and practices vary between businesses. [28]

When adding information in invoice coding, users are documenting the information for accounting purposes to coding rows. Each coding row uses multiple dimensions to record accounting information for different purposes. Coding dimensions describing the financial accounting information are usually *account code* and *tax code*, and those are filled to almost every invoice across all businesses.

Account code is used to structure and categorize similar spending together in the accounting process. In other words, the account code categorizes what was bought. The different account codes define the account chart of the business. Many companies have adopted a hierarchical schema for account chart where the general class is presented by the first number and granularity of the classification increases towards the end of the code. Assigning tax codes is a way to categorize the spend into different taxation classes to cope with tax regulations. [25]

Other coding dimensions are used for management accounting purposes to keep track of the spend. Some commonly used dimensions include *cost center code*, which determines

the entity responsible for the spend, *project code* which assigns spend to a certain project, and *internal order* which classifies the location of spend. Unlike dimensions for financial accounting, management accounting dimensions can be left empty, because those are used only for internal purposes.

When an invoice has multiple items, the items can be divided into multiple rows as seen in the example invoice in Figure 1.1. If the items share the same coding values, only one coding row is created for the invoice. However, different items might need a different coding value and it is then necessary to input multiple coding rows. It is also possible that for example, two cost centers share the spend, so even an invoice with one line item can have multiple coding rows.

These practices lead to a situation where the number of rows in the invoice and the number of coding rows are not strongly connected. In theory gross and tax sums assigned to coding rows should sum up to the total gross and tax sum of the invoice header, but in practice rounding and special cases cause mismatches. Construction of the correct coding row for each invoice line is therefore an ill-posed problem.

Business also takes actions to make the invoice coding easier and deterministic. For example, in order to assign costs correctly the person ordering may ask the supplier to present the responsible cost center, project, or internal order in the invoice when making the order. Similarly, the supplier writes the PO number to the invoice when available. This is a best practice that is used by some organizations for some invoices.

When coding value is presented on the invoice it is trivial for an AP clerk or business user to determine the dimension value the cost center or project code correctly by looking at the invoice. Sometimes, not the actual value of the dimension is in the invoice, but a value that is strongly related to some dimension value, like a person's name, address, or name of the unit. These named entities help to find the correct dimension value when some additional information about the organization is known or rules are learned from the earlier invoices.

Table 2.1. *Commonly using coding dimensions. The "Value in invoice?" field indicates if correct value for coding dimension can be read from the invoice content.*

Dimension name	Why?	Who knows the correct coding?	Value in invoice?
Account code	Financial reporting	AP clerk	No
Tax code	Tax compliance	AP clerk	No
Cost center code	For management	Business user	Ideally yes, but not always
Project code	For management	Business user	Ideally yes, but not always
Internal order	For management	Business user	Ideally yes, but not always

The commonly used coding dimensions introduced are collected in Table 2.1. The table shows why coding dimension is used, who has the knowledge of the correct coding and

can the correct coding values are in the invoice content already.

2.1.2 Determining the correct coding

By taking a look into the definition of coding dimensions, some intuition of how coding is done can be found. Account code categorizes the spend so the category of the items with knowledge of the used account chart gives a starting point. Tax percentages of different taxation classes vary, so looking at the tax percentage of items with country information will hint at the correct tax code along with knowledge of what was bought. The compensability of taxes also varies across countries, so knowing the local rules helps to determine the tax code. For cost center and project codes, the process is a bit different. Customer reference, invoice delivery address, etc. might reveal the cost center or project responsible for spend. As mentioned in the previous Section 2.1.1, the correct cost center, project code, or internal order might also be added to invoice fields.

Also looking at the assumptions made in earlier attempts to automate coding tells about how coding is done. Earlier, Basware was using a naive classifier that copied the coding row from the latest invoice with the same supplier and receiving organization. The fact that the system was implemented and is still working for some customers tells that the assumption is somewhat valid. It implies that the supplier and receiving organization are important features for classifying all the coding dimensions. Similar naive classifiers have also been implemented by other organizations [3].

Table 2.2. Example coding row for example invoice presented in Figure 1.1.

row #	Gross sum	Tax sum	Account code	Account name	Tax code	Cost center code	Cost center name
1	1440.00	345.60	68521	Research and development expenses	FI24	6013	Product development Tampere
2	99.99	24.00	68531	Software licenses	FI24	6013	Product development Tampere

Table 2.2 presents example coding rows for the invoice in Figure 1.1. There are 2 coding rows because the items in the invoice are coded to 2 different account codes. Other coding dimensions are shared between the two rows. This invoice is an example of the case that cost center is specified in the invoice as reference information and the value is found and copied from there to coding rows.

2.1.3 AP automation

This chapter introduces invoice automation techniques that try to reduce work for invoice coding and reviewing. These are the traditional and well-established techniques and high level of automation can be reached when configured and used correctly.

When orders are made systematically in ERP or a separate purchase system, the PO

is in a defined structured format. Therefore, the invoice for the purchased items can be automatically matched to the corresponding PO. A third document, a list of supplied items, called goods receipt, filled by the buyer, can also be matched to the invoice and PO to verify the items are what was ordered. In AP context, *matching* is the process of finding matching PO, invoice, and optionally goods receipt documents with the goal of linking those together [25, ch. 1]. To streamline the process, coding and approving can be made in the ordering phase and not repeated when the invoice is received.

Another way to streamline the AP process is to have predefined rules for recurring invoices called spend plans. With spend plans, invoices from the same supplier and reference have automatic processes that need to be made, coded, and approved only once. Also, invoice dates can be used to match the new invoice to existing spend plans. Limits for the maximum amount of price can be set to trigger manual intervention for invoices deviating from the original plan. In consumer context, automatically approved e-invoices are the equivalent for spend plans.

Matching and spend plans that are well-established and deterministic ways to automate AP tasks are visualized in Figure 2.3.

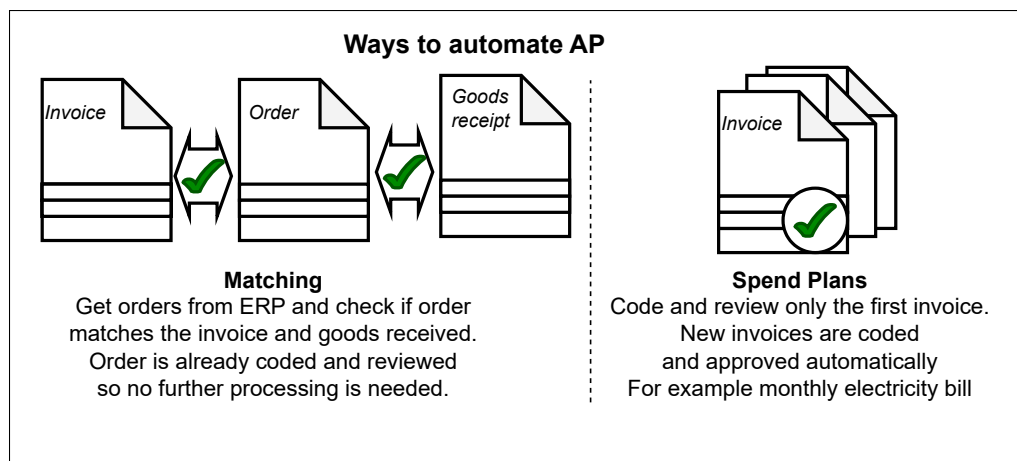


Figure 2.3. Ways to automate the AP tasks.

Parts of the AP process presented in Figure 2.2 can be automated even if a matching PO or spend plan is not found. Usually, the low-hanging fruits in terms of automation are the things that can be easily automated by coming up with heuristics rules. For example the validation of the received invoice in step 1 of Figure 2.2 contains checks like is the supplier's bank account correct. This can be easily automatically checked using heuristics and supplier data that is saved to the system containing the bank account. Another simple example is the resolving of the recipient, meaning who should approve the invoice. The correct approver can in some cases be found from the organizational chart or from the list of managers for the assigned cost center.

2.2 Receiving invoices

Suppliers create and send invoices in various processes and systems and use diverse formats. Therefore, the invoice data received by the AP department needs to be unified to process within one AP system. In the scope of this work, it is important to gain an understanding of where the invoice data comes from to identify the restrictions of used data sources and the processing steps.

Figure 2.4 presents some of the possible invoices origins that arrive in the system. A digitalization processor that extracts the data from invoice images or PDFs and a connector that converts digitized invoices and different e-invoice types to standardized TEAPPS format. TEAPPS is the format that Basware's AP software accepts as input, but other AP softwares may use different standardized formats.

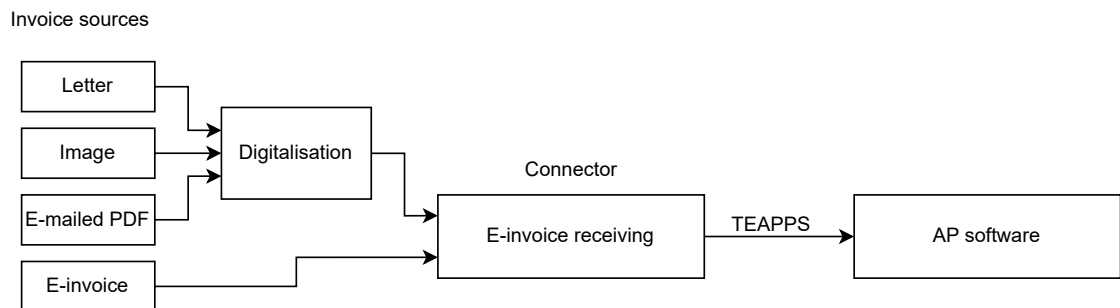


Figure 2.4. Receiving different types of invoices. Non-e-invoices need to go through a separate digitization process to extract the information into digital format. The connector unifies different e-invoicing standards to the TEAPPS format that is fed into the AP software. The original digitized invoice image is saved for human intervention.

For e-invoicing, multiple standards to format the invoice data exist. In the European Union, all e-invoices must follow the semantic schema that is defined in EN 16931 standard [7]. Various implementations of the European standard exist and are used commonly at the country level for historical reasons. For example in Finland companies are using e-invoicing standards TEAPPSXML 3.0 and Finvoice 3.0 that are able to comply with the European standard and add some special information on top [22].

E-invoicing standards are usually implemented using XML format, where standards define the structure of the document and some mandatory fields, but leave the door open for various free text fields to make custom invoices and even another type of documents. The standards define massive amounts of optional fields, and only a subset of fields are used by a single invoice. From the e-invoice XML, an invoice image can be generated for human intervention using defined templates. The e-invoice XML is typically richer in information than the generated image, because it contains all the metadata of the invoice.

The different e-invoicing standards are generally not compatible with each other, because the structure of the document is different, and also alternative names and syntax for spec-

ifying the elements are used. However, to process all the received invoices within one AP software a unified format for invoices is needed. In Basware's AP system, all the invoices are transformed to custom TEAPPS format which is an extension of TEAPPSXML.

To transform other e-invoicing standards to TEAPPS or any other format, a set of human expert made data mappers are used to map the data from foreign fields to TEAPPS fields. Some e-invoicing standards are close to each other and the generation of mapping is therefore easier whereas others differ significantly, which causes loss of data and other inaccuracies in the transformation. If the target is just to process the invoice within the AP system, only a subset like the header information of the e-invoice can be mapped to simplify the conversion process.

The original XML document of the invoice presented in Figure 1.1 is presented in Finvoice format in Appendix A. Note that conversion in this case is done manually, and might not be accurate. Still, it shows how the invoice image and XML relate.

For non-e-invoices meaning the PDF, email, or paper invoices digitisation is needed to transform the data into TEAPPS format. The information captured is typically restricted only to data that is visible on the invoice and is therefore missing some of the metadata like e-invoicing endpoint and routing. The text from the invoice can be captured in digitized format by using an optical character recognition (OCR) system. In the case of machine-readable files, the text can be just trivially copied from the invoice.

The challenging part is again similar to e-invoices of how to map the found text into the correct fields of the TEAPPS structure. Again human experts can make rule-based templates by looking at the texts and the positions of the texts. Also, advanced techniques leveraging machine learning are proposed for this mapping problem [23]. Like converting e-invoicing standards, only a subset of the invoice information can be mapped to simplify the process.

The drawback of converting invoices to a unified format is the loss of data in the transformation process. Whether it is due to limits of the mapping or faults in the digitalization process. For example, if triggering of the automatic matching described in Section 2.1.3 relies purely on a standardized e-invoice field that should include the order number and nothing more, manual intervention is needed in case of any failures due to the order number being in a different field or presented with unwanted prefixes like "*order number:*". The same goes for all the other fields in e-invoices and the processing of the transformed data needs to therefore be robust to missing fields or ask the user to capture needed missing values.

The e-invoicing standards do not specify a certain field to carry the coding information unlike for order information. Therefore, even if the buyer asks the supplier to add the cost center code to the invoice, there are many possible fields in which the information can be

added. For that reason, it is impossible to create robustly working mappings for capturing coding values like cost center codes from invoices.

From the unified TEAPPS format invoice header information is extracted to the database as it is often enough to use the header in further processing. The header fields are generally well-defined in all of the invoices and are the minimum fields required in financial accounting. The header only includes the information relevant to the whole invoice such as but not limited to supplier, targeted organization, net, tax, and total sums.

The whole TEAPPS is saved as an attachment and is accessible if needed, but the data varies significantly between invoices and there is no point in feeding all the TEAPPS fields to a relational database. For some invoices, non-header data is extracted from TEAPPS to the database to for example enable processing of the individual invoice lines. The extraction is again done with human made rule based mappings, which work well for well-defined invoices but fail in case of any exceptions.

2.3 Related works

Automating coding row creation can be seen as a rather narrow application field that still has substantial commercial potential. Therefore, only a few academic contributions exist, but many products tackling the problem are available to reduce the amount of manual work needed.

Few studies investigate the application of machine learning to solve account code classification problem. Bardelli et al. [1] considered account and tax code prediction on e-invoices from Italian companies. The authors compared the performance of different classification methods for both sent and received invoices with different feature extraction methods from invoice line descriptions. They also proposed a solution to the problem where the number of invoice lines mismatches the number of coding rows and therefore ground truth coding data is missing for some invoice lines. They showed that formulating the problem as multi knapsack problem known in combinatorial optimization was able to assign line-level labels to 61% of lines. The used dataset consists of anonymized invoices from 2 real-world accounting firms summing up to around 35 thousand invoices per firm in 15 months. Based on the invoice counts and size of account charts it is likely that the data consists of invoices from multiple small companies.

Classification algorithms compared by Bardelli et al. [1] were Random Forest, AdaBoost (tree-based ensemble model), and Multi Layer Perceptron which all suited well for multi-class classification problem. Features from invoice lines were extracted with Bag of Words and Word2Vec algorithms. They concluded that the highest accuracy was obtained for account codes with Random Forest classifier with Word2Vec features and for tax codes with AdaBoost and Word2Vec.

Munoz et al. [21] built on top of the work of Bardelli [1] by introducing a hierarchical classification method for account codes. They also explored more recent methods for feature extraction from line-level descriptions namely large language model (LLM) embeddings from BERT (Bidirectional Encoder Representations from Transformers). Usage of standard pre-trained BERT proved not to be beneficial over traditional methods like Bag of Words or Word2Vec, but fine-tuning BERT with invoice data was found to give a significant boost in performance.

They concluded that the use of hierarchical neural network architecture performs better than classification methods used in previous research and that using actual bank accounts as a hierarchy proved to be better than the initial idea of using the hierarchy of account codes itself. The dataset was extracted from Xero¹, which is a cloud-based accounting platform used by small and medium-sized businesses in Australia and New Zealand.

Three master theses from Swedish universities have researched the classification problem of account codes [3, 4, 12]. Bergdorf [4] showed that Random Forest gives the best classification performance among support vector machine and naive classifier using simple rules, but concluded that the best approach would be to combine rule-based methods with machine learning-based methods. Hedberg [12] ended up in similar findings when researching classification models for account and cost center code suggestions. The author also introduced filters for suggestions using supplier commonality, precision for the supplier in training, and a number of invoices by the supplier in the evaluation set. Lessner [16] and Esswein [9] studied a recommender system approach to account code suggestion to overcome the cold start problem of classification algorithms trained for each organization.

2.4 SmartCoding - Basware's solution to coding suggestions

To give coding proposals for users SmartCoding utilizes the invoice header data and coding information of manually coded invoices that are saved to a database. A classification model for each coding dimension used by a specific customer is trained using supervised learning. The used header data does not include information from invoice lines, but only the information relevant to the whole invoice such as but not limited to supplier, targeted organization, and net, tax, and total sums. An example invoice with highlighted header data can be seen in Figure 1.1.

Because data is restricted to header values, it is difficult to predict multiple coding rows as those are often needed when there are multiple invoice lines with varied coding information. Therefore, current SmartCoding proposes only the first coding for an invoice. Also,

¹<https://www.xero.com/>

the training data is restricted to contain only the first coding rows. Most of the invoices in the scope of automatic coding row suggestion have only one coding row as will be seen later in Chapter 4.

The solution consists of two phases, first the customer data is analyzed using custom analyzing tools, and the optimal configuration is created for each customer. After the analysis, models are automatically retrained every day with new data added to the training dataset. Coding proposals are created for each new invoice that arrives in the system and the most probable value for each dimension is preassigned to the coding row. In addition to that, if the user chooses to edit the coding value the 5 most probable values are presented in the dropdown menu. An example of SmartCoding predictions is Basware's AP software user interface is shown in Figure 2.5.

Account Code		Cost Center Code		Tax Code	
1880, Product Marketing	78%	1060, Marketing	49%	S06	91%
1200, Software purchases	9%	1050, Advertising	44%	S24	4%
1220, Printing purchases	4%	1140, Quality Assurance	2%	S0	<1%
1550, Learning Supplies	2%	1090, Accounting	<1%	S05	<1%
1570, Internal events and meetings	<1%	1160, Presales	<1%	S04	<1%

Figure 2.5. Example proposals of SmartCoding as presented to the user. Percentages indicate the model confidence for each prediction. By default, the first prediction is assigned to the coding row.

As seen in Section 2.1 some coding dimensions have the correct coding value or related named entity available in the invoice. For that, SmartCoding has an expansion called Intelligent Capture (IC). IC is a computational model that searches the already-used dimension values from the invoice TEAPPS structure and suggests the found dimension values for coding. Each coding dimension can use the classification model, IC model, or both, in which case the IC prediction is prioritized.

3. METHODS

This chapter takes a theoretical overview of the used machine learning techniques. First, the overall architecture of the machine learning pipeline is described to understand how invoice data is transformed into coding predictions. Training procedure of the system using human-coded invoices is also formulated.

Next, the detailed steps of the machine learning pipeline are discussed including feature extraction and classification. Feature extraction converts the invoices into a numerical form that is accepted by the classifier. Many known methods for feature extraction from unstructured textual data exist and the relevant ones for this work are introduced. Classification is performed over the extracted features from invoices to predict the most probable coding values among the pool of existing coding values. In this work, the used classification model is a decision tree based XGBoost, which efficiently captures the patterns in the data.

Finally, evaluation methods are discussed. Evaluation is done to determine how well the proposed machine learning pipeline works. It is important to address the system performance by looking at the predictions manually but defining evaluation metrics allows the comparison to other methods, hyper-parameters, or datasets by summarizing the performance to one number over many invoices that is comparable to other methods.

After this chapter, the reader should have an idea of how the used methods work and have the knowledge to understand the results of the experiments presented in the next chapter.

3.1 Machine learning approach

To predict the coding value for each dimension of the invoice using machine learning, human-coded invoices can be used as training data. This is called supervised learning as the system learns general rules of coding from labelled examples. As there are multiple possible values for coding dimensions a multi-class classification is needed. Multi-class classification means that the correct value needs to be selected among a pool of values, rather than just binary true or false. Still, only one value is ultimately selected. In traditional natural language processing (NLP) systems, the process of classifying text

documents is usually divided into two main sections: feature extraction and classification [14].

The first part of the system is a feature extractor that takes the raw text and processes it to vectors of numbers as machine learning classifiers cannot understand the raw text. One possible way to transform text into numbers is to present each character as a number using for example ASCII encoding. More recent methods like tokenization take the statistical approach and iteratively combine the frequent sequences of characters together and present the sequence as one number [26].

Selecting the feature extraction method, also known as feature engineering, can have a great impact on the overall performance of the system. Recent methods like deep neural networks often take the end-to-end learning approach and try to minimize the needed feature engineering work, but also comes with costs of more data and computing power required. More methods for feature extraction from unstructured textual data are discussed in detail in the following chapters as it is a key contribution of this thesis.

The second part of the system performs the classification of the extracted feature vectors. In supervised learning, the classifier is first trained using labeled data, where the goal is to minimize the error between the classifier outputs and the ground truth labels by forming a generalizable model. A multi-class classifier takes in the features extracted previously and assigns a probability for each known class. Basically, any classification method known in machine learning can be used to classify text documents. Some commonly used methods include linear regression, support vector machines (SVM), decision trees, random forests, and neural networks [14].

Based on observations previously, this work proposes a method to improve the earlier coding prediction system by utilizing the unstructured invoice content using a selected feature extraction method. The high level architecture of the system is presented in Figure 3.1. The proposed pipeline is used individually for each coding dimension used by the specific customer. So to give coding predictions to N dimensions and M customers, $N \times M$ classification models are trained. The model for each dimension and customer may utilize different inputs depending on the feature extraction technique used and values selected from header data. The output of each model is a vector of probabilities for all known values for that coding dimension.

Figure 3.1 also presents the multi-class classification output, with highlighted best proposal. In the classification output table left column presents all the possible coding values for one dimension and the right column the confidences that the classifier assigns to each coding value. The most probable proposal according to the classifier is the one with the highest confidence.

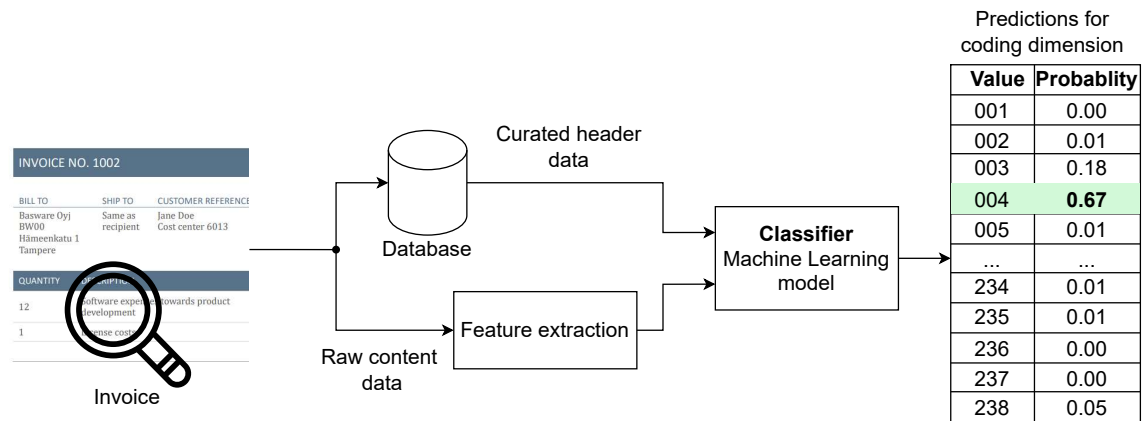


Figure 3.1. Overview of proposed machine learning pipeline. The proposed dimension value (004) with the highest probability (0.67) is highlighted in green. Note that this figure presents only predictions for one invoice and one coding dimension.

3.2 Feature extraction from invoice content

Features from the invoice content are used in this work to gain an advantage over the old system that used just the header data. The data presenting the invoice content is the TEAPPS that is described in more detail in Section 2.2. Technically, the TEAPPS is semi-structured data, that contains structure and tags holding the actual payload. However, the amount of variation in TEAPPS structure among invoices even within one company is so great that the TEAPPS data is treated as unstructured data.

TEAPPS data contains metadata about the invoice like routing information and unique IDs that are not usable for feature extraction. By observing the invoices, it was found that the metadata is in selected tags of the XML. These tags were identified and rules to filter out the tags containing the metadata were introduced. Selecting only a subset of the tags means that the data is handled as semi-structured data in the preprocessing phase because information about the structure is used to filter out the irrelevant tags. By filtering the tags the textual content of the TEAPPS data is closer to the words extracted from the invoice image that does not include metadata either.

After relevant TEAPPS tags have been selected the payload of the XML document is concatenated to one long string. This is the invoice content data in a textual format that is treated as unstructured data in the following steps. The input and output of the conversion of e-invoice to text is illustrated in Figure 3.2. In that case, there are no tags to be filtered before the concatenation of values.

As seen in Section 2.2 the TEAPPS data has gone through a standardization process that causes a loss of data. Treating the invoice content as unstructured text allows the proposed machine learning pipeline to use also other data sources than the TEAPPS. For example in the case of PDF invoices, it would be beneficial to use the raw text content

```

47     <BuyerTownName>Tampere</BuyerTownName>
48     <BuyerPostCodeIdentifier>33100</BuyerPostCodeIdentifier>
49     <CountryCode>FI</CountryCode>
50     <CountryName>Suomi</CountryName>
51 </BuyerPostalAddressDetails>
52 <BuyerContactPersonName>Jane Doe</BuyerContactPersonName>
53 <BuyerContactFreeText>Cost center: 6013</BuyerContactFreeText>
54 </BuyerPartyDetails>
55 <InvoiceDetails>
56 <InvoiceTypeCode>INV01</InvoiceTypeCode>
57 <InvoiceTypeCodeUN>380</InvoiceTypeCodeUN>
58 <InvoiceTypeText>Invoice</InvoiceTypeText>
59 <OriginCode>Original</OriginCode>
60 <InvoiceNumber>1002</InvoiceNumber>
61 <InvoiceDate Format="CCYYMMDD">20230201</InvoiceDate>

```



```

"Tampere 33100 FI50 Suomi Jane Doe
Cost center: 6013 INV01 380 Invoice
Original 1002 20230201"

```

Figure 3.2. Example of how invoice XML (Finvoice from Appendix A) is transformed into string by concatenating the values.

from the invoice rather than TEAPPS which contains only parts of the full text. Similarly, conversion of e-invoices to TEAPPS can lose some of the text in the invoice and it would then be beneficial to use the raw payload as a text instead.

As stated before human-readable text needs to be further processed into a sequence of numbers for machine learning purposes in a process called feature extraction. Finding efficient features from textual data for different NLP tasks has been under comprehensive study [14]. Therefore, many methods exist and each proves potential for various tasks.

In the problem of coding prediction, Bag of Words (BoW), Word2Vec, and BERT feature extraction methods have previously been studied to utilize the text from the invoice [1, 21]. This work studies and tests these suggested methods, excluding BERT, for feature extraction. Additionally, latent semantic analysis (LSA) [8] and FastText [5] feature extraction methods are considered as an improvement to the suggested methods. These methods are built on top of BoW and Word2Vec respectively and have been found to work well for feature extraction tasks [14].

Using LLM embeddings like Munoz et al. [21] used pretrained BERT for feature extraction was not considered in this work, although fine tuning the embedding with invoice data gave the best performance in their work. The simplistic nature of the text in the invoices hinted that using a more complex model like LLM would not be beneficial as the task is more to utilize information from single words. Fine tuning is also considered a demanding and time consuming task and therefore experimenting fine tuned LLM embeddings was not considered in this work.

There exist many other feature extraction techniques like linear discriminant analysis (LDA), hashing vectorizer, tokenization, or part-of-speech (POS) tagging [14]. These methods were not identified as potential as selected methods in previous works on coding prediction and overall do not fit the characteristics of the datasets. Therefore, no further study towards these methods were conducted.

3.2.1 Bag of Words

Bag of Words (BoW) is a simple way to form a feature vector from text. The basic idea of BoW is to present each document as a vector of its word occurrences. That means that for each document, BoW counts how many times each distinct word is used. An example of BoW feature extraction is presented in Table 3.1.

Document text	the	cat	dog	jumps	lazy	...
The lazy cat sleeps.	1	1	0	0	1	...
The brown dog barks.	1	0	1	0	0	...
Cat jumps and jumps.	0	1	0	2	0	...

Table 3.1. Bag of Words representation for 3 documents.

The drawback of the simple BoW feature extraction is that the feature space grows linearly with the size of the vocabulary for a set of documents. The feature space can also be sparse, which means that most of the distinct words appear only in few documents. Additionally, BoW feature extraction loses the information of the word order and can not, therefore, make a distinction between for example phrases *"this is good"* and *"is this good"*.

The features extracted by BoW can be further enhanced by calculating the Term Frequency Inverse Document Frequency (TF-IDF) values for each word occurrence in the BoW matrix. The formula for calculation the TF-IDF weight $W(d, t)$ is

$$W(d, t) = \text{TF}(d, t) \log \left(\frac{|D|}{\text{DF}(t)} \right), \quad (3.1)$$

where $\text{TF}(d, t)$ is term frequency calculated already in BoW matrix, $|D|$ is number of documents and $\text{DF}(t)$ is number of documents containing term t [14]. TF-IDF tries to reduce the impact of words that are common to many documents but still suffers from the same drawbacks as basic BoW.

The definition of a word in BoW feature extraction can be extended from space separated words. To overcome the problem of losing the word order, N-grams of words can be fed to BoW as words. Also, character N-grams can be used to tackle the problem of inflections or typos. N-gram is a contiguous sequence of N words or characters. The example of forming N-grams from a sentence is presented in Table 3.2. The BoW feature extraction can use a range of N-grams like N from 1 to 3 as features and calculate the occurrence of each N-gram for documents.

To limit the size of the feature space, only a subset of distinct words from BoW can be selected by pruning. Words can be pruned using document frequency which calculates how many documents have that word occurring. From the BoW matrix, it is trivial to calculate

Unigrams (N=1)	Bigrams (N=2)	Trigrams (N=3)
The	The cat	The cat sat
cat	cat sat	cat sat on
sat	sat on	sat on the
on	on the	on the mat
the	the mat	
mat		

Table 3.2. *N*-grams ($N \in \{1, 2, 3\}$) for sentence "The cat sat on the mat".

the document frequency by counting the non-zero elements for each word. Words with the highest document frequency are usually common words in language like *and*, *the*, *a*, or *an* that do not provide any valuable information for classification and can be safely pruned. Also, words with low document frequency are usually pruned. In multi-class classification setting, vocabulary pruning might affect the performance of infrequent classes as the relevant words for those might be pruned.

On top of pruning feature space with document frequency, only the top K words with the highest document frequency can be selected. Another more interesting option is to find the words that have the highest correlation with the target variable. To calculate the correlation of word occurrences and the target dimension values a categorical correlation calculation like Theil's U [29] can be leveraged. Theil's U correlation $U(X|Y)$ also known as the uncertainty coefficient can be calculated from the entropy $H(X)$ and conditional entropy $H(X|Y)$ of the distributions of target variable X and word Y :

$$H(X) = - \sum_x P_X(x) \log(P_X(x)) \quad (3.2a)$$

$$H(X|Y) = - \sum_{x,y} P_{X,Y}(x,y) \log(P_{X|Y}(x|y)) \quad (3.2b)$$

$$U(X|Y) = \frac{H(X) - H(X|Y)}{H(X)}, \quad (3.2c)$$

where $P_{X|Y}(x|y)$ is the conditional distribution of X given Y .

3.2.2 Latent semantic analysis

One option to reduce the size of the feature space of BoW is to compress the information of words to lower dimensionality. Well-known matrix dimensionality reduction techniques like Principal Component Analysis (PCA) or Singular Value Decomposition (SVD) have been successfully used in NLP tasks for feature extraction [14].

Particularly, latent semantic analysis (LSA) has proven potential for finding semantic sim-

ilarities in text documents. The idea of LSA is to collect common factors from the word occurrence matrix that represents the underlying semantic structure of the set of documents. An assumption here is that words with similar meanings will occur in similar documents. The goal is to find semantic meaning that describe the document in higher level than just the words using statistical methods. Therefore, the different words in the documents will be associated together based on the context they occur using high level concepts. [8]

Originally LSA was proposed as retrieval engine called latent semantic indexing (LSI). Previous retrieval systems failed in cases where user was not able to describe the search using the same terms as the wanted document. Using semantic similarity of the terms, the authors where able to retrieve documents with semantically similar meaning not limited to searching of the same words. The retrieval can be done simply by measuring the similarity of the documents in the semantic space, because semantically similar documents will end up close to each other in LSA transformation. [8]

The drawbacks of LSA are that the features are no longer easily interpretable as LSA forms the semantic concepts by taking massive linear combination of all the used words in the word matrix. LSA features can not make a distinction between polysemies meaning words with many meanings and can not capture phrases longer than the representation of a word in the underlying BoW model. Also, an extra hyperparameter for the target number of dimensions needs to be selected.

The word occurrence matrix in LSA is BoW or TF-IDF as presented in the previous chapter. The word occurrence matrix is compressed using a truncated variant of SVD. Truncated SVD is a method to find a low-rank decomposition of the word matrix. The normal SVD is a method to decompose the word matrix into its singular values Σ and singular vectors of words \mathbf{T} and documents \mathbf{D} so that the original word occurrence matrix \mathbf{X} can be presented as

$$\mathbf{X} = \mathbf{T}\Sigma\mathbf{D}^T \quad (3.3)$$

[8].

By definition, \mathbf{T} and \mathbf{D} are orthogonal matrices meaning that the columns are linearly independent and Σ is a diagonal matrix, where the diagonal contains the singular values of \mathbf{X} . If the matrix \mathbf{X} has a shape of $t \times d$ (note, transpose of the described BoW matrix) \mathbf{T} will have a shape of $t \times t$, \mathbf{D} shape of $d \times d$ and Σ shape of $t \times d$.

A low-rank approximation of \mathbf{X} , $\hat{\mathbf{X}}$ in the truncated version of SVD takes only the k largest singular values from Σ to reduce the feature dimensionality to k . By sorting the singular values in Σ and corresponding singular vectors in \mathbf{T} and \mathbf{D} , and taking the k largest values, 3 new matrices \mathbf{T}_k , \mathbf{D}_k and Σ_k are formed. The new matrices approximate the original matrix \mathbf{X} by expressing it as a combination of the most important components. By

definition, this is the most optimal way of presenting the matrix \mathbf{X} with rank of k . Formally,

$$\mathbf{X} \approx \hat{\mathbf{X}} = \mathbf{T}_k \Sigma_k \mathbf{D}_k^T \quad (3.4)$$

[30]. Figure 3.3 visually illustrates the involved matrices with assumption that elements of the matrices are sorted by the singular values in Σ .

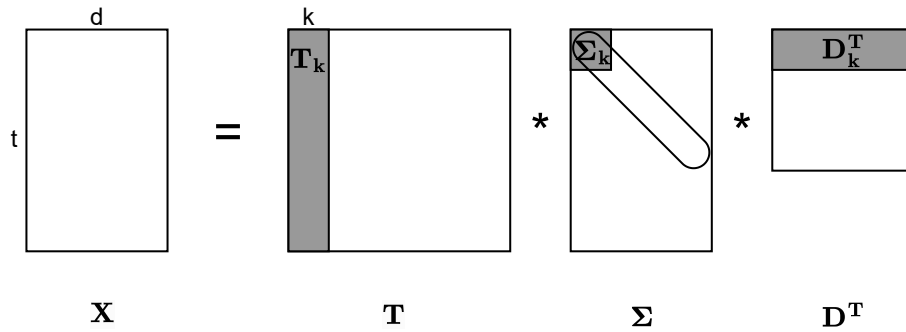


Figure 3.3. Singular value decomposition of term-document matrix \mathbf{X} . Highlighted in grey are the submatrices of \mathbf{T} , Σ and \mathbf{D}^T used in truncated version of SVD to get the rank k approximation $\hat{\mathbf{X}}$ of \mathbf{X} . Figure adapted from [30].

In the problem of reducing the feature dimensionality, finding representation of \mathbf{X} where $k < t$, only \mathbf{T}_k is needed. \mathbf{T}_k presents every column of \mathbf{X} (documents) as a linear combination of the columns of \mathbf{T}_k , with the weights coming from columns of $\Sigma_k \mathbf{D}^T$. Therefore, truncated SVD can be first fitted with some documents \mathbf{X}_1 to get the \mathbf{T}_k , and then another set of documents \mathbf{X}_2 can be transformed to the same feature space by taking matrix multiplication $\mathbf{T}_k^T \mathbf{X}_2$. [30]

The SVD computation in LSA is the same as doing dimensionality reduction with PCA. In PCA matrix \mathbf{X} is preprocessed to have zero mean for each row and then eigenvectors of covariance $\mathbf{X}^T \mathbf{X}$ matrix that have the largest eigenvalues are found. This results in the same result as finding the singular vectors with SVD, however, the normalization can affect the absolute values. [30]

Efficient numerical methods to calculate the truncated SVD exist in literature [11] utilizing iterative randomization techniques. The details of these computational techniques are not described in this work.

3.2.3 Word2Vec

A word vector, also known as word embedding, is a numerical representation of a word in high dimensional space. In this high dimensional space, each word is presented with a continuous valued vector so that words with similar meanings will be close together in the space. The core idea of word vectors is to capture semantic and syntactic relationships

between words, a bit like in the previously introduced LSA method.

With techniques like Word2Vec [20], FastText [5] or GloVe [24] it is possible to learn efficient word vectors from large unlabelled text data that are able to capture multiple degrees of similarity among words. These neural network based methods surpass the performance of previous methods when comparing the word relations.

This chapter introduces the Word2Vec approach and the next one builds on top of it and introduces the benefits of FastText method over Word2Vec as those are the techniques used in this work.

Word2Vec, published by Mikolov et al in 2013, builds on top of the observation that neural network based language models form internal representation for word vectors when trained for language modeling task [20]. One of the early architectures of neural network language models was introduced by Bengio et al. in 2003 [2]. Their proposed architecture consists of an embedding layer that transforms the word indexes into word vectors, one hidden layer, and an output layer with softmax activation. The model was trained for next word prediction over a dataset of 15 million words, which was large at the standards of the time. The work of Mikolov took the idea from these neural network language models but proposed an architecture that would generate the best possible word vectors while reducing the computational complexity.

The approach in Word2Vec consists of a shallow neural network, that has only the embedding layer and output layer. That way the amount of trainable parameters is minimal, which reduces the computational complexity and the model is easier to train for large amounts of data. A finite context length, meaning the number of neighboring words the model is looking at when learning the word vector, needs to be set and authors found out that using the previous and following words gave the best performance. They proposed two distinct architectures, continuous bag of words (CBOW) and skip-gram, for word vector models presented in Figure 3.4.

To gain a deeper understanding of the model, the mathematical details of the CBOW architecture can be presented as follows. First, the vocabulary is tokenized meaning that it is transformed into indexes, where each distinct word is presented with a running number from 1 to V . Then, each word in the context is embedded in N dimensional space by taking corresponding rows from embedding matrix \mathbf{C} sized $V \times N$, that holds the actual word vectors.

The word vectors of the context are then summed to form one continuous bag of word representation that loses the information of the word order in the context. The CBOW vector is then projected back to vocabulary sized vector by using projection matrix \mathbf{P} sized $N \times V$ forming the output.

To calculate the loss function for training, softmax of the output is taken and negative log

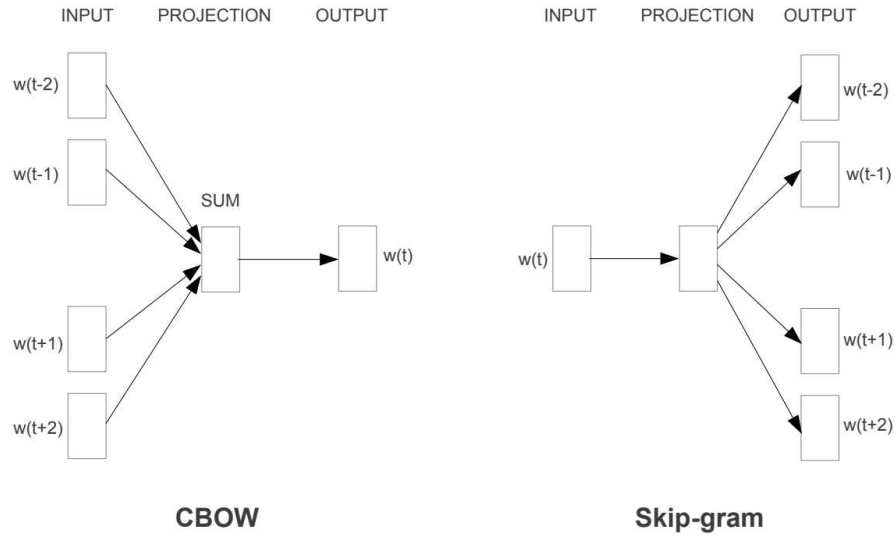


Figure 3.4. Variants of the Word2Vec architecture, as illustrated in [20]. Left: continuous bag of words (CBOW) that is trained to predict the $w(t)$ based on the context. Right: Skip-gram, that is trained to predict to context words $w(t-2)$ to $w(t+2)$ based on $w(t)$.

likelihood loss is calculated by taking the index of the correct word $w(t)$. Formally, to predict word $w(t)$ and calculate the loss, elements of the CBOW vector \mathbf{A} are calculated

$$\mathbf{A}_i = \sum_j^S \mathbf{C}_i(w(t-j)), \quad (3.5)$$

where S is the context, for example $S \in (-2, -1, 1, 2)$. Then, prediction is given by matrix multiplication

$$\mathbf{y} = \text{softmax}(\mathbf{AP}), \quad (3.6)$$

that is transformed to probability distribution using softmax. Finally, negative loss likelihood loss is calculated

$$\text{loss} = -\log \mathbf{y}(w(t)). \quad (3.7)$$

The models are trained by using stochastic gradient descent and backpropagation by minimizing the negative log likelihood. The details of these well know neural network optimization techniques are not described in this work but can be studied from literature [10]. Mikolov et al presented further tricks and enhancements to their skip-gram model in a follow up paper [19] including hierarchical softmax, negative sampling methods, and subsampling of frequent words.

One somewhat surprising capability of the word vectors is to find relations from the text and to perform queries of these relations by simple algebra. For example, Mikolov et al. [20] tested the trained Word2Vec model's ability to find relations for word pairs. For example, *what is for Italy the equivalent that Paris is for France*. And the answer from the model was Rome. Similarly, the model learned that Tokyo has an equal relationship to

Japan.

These queries were just obtained by calculating $vector(Paris) - vector(France) + vector(Italy)$ and looking for similar vectors of the resulting vector. Similarity of vectors can be calculated by using for example cosine distance. Furthermore, they observed also syntactic similarities like $vector(bigger) - vector(big) + vector(cold)$ equals something close to $colder$. Some observed relations of words are presented in Figure 3.5.

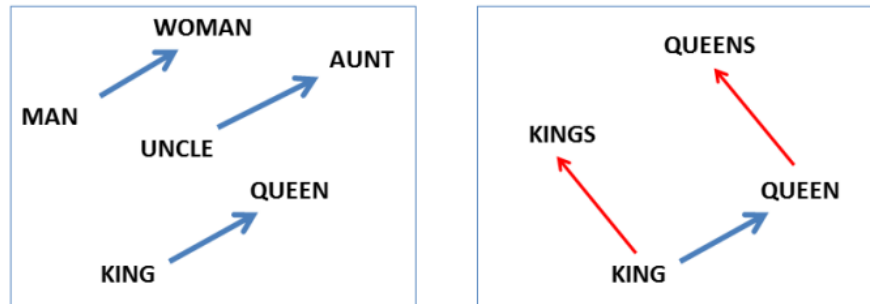


Figure 3.5. Relation of word vectors as presented in [18]. Left shows a projection of three word pairs illustrating the gender relation. Right shows a different projection and plural relation for a word pair. In word vector arithmetics the idea is that the vector presenting the relation is the same. In high dimensional space, multiple relations exist for each word.

In the scope of this work, a representation of the whole text from one invoice is needed to get the feature vector for an invoice. The used approach in this work was to train Word2Vec model using the text extracted from the invoices and then calculate word vectors for each word in each invoice. Then, words within one invoice are fused into one feature vector by taking a sum of the word vectors. The sum therefore has the same dimensionality as obtained word vectors. Another option is to weight the word vectors before summing with TF-IDF term, presented in Section 3.2.1. Also, more specific techniques to calculate vector representation of documents like Doc2Vec [15] exists.

The main drawback of Word2Vec models is that learning high performing high dimensional vectors needs a large amount of training data. The model lacks contextual awareness as the context window is usually short and it loses the meaning of word order. Learning vector representation for rare words that occur only a few times in the training data is challenging for Word2Vec models. The model is generally limited only to words that it has seen from the training set, thus fails to produce a word vector for any out of vocabulary word. In the scope of this work limit of Word2Vec is that there are no well defined ways to form document or sentence vectors by combining the vectors of the words.

3.2.4 FastText

To address the Word2Vec drawbacks, mainly the problem of out of vocabulary words, Bojanowski et al [5] proposed the FastText architecture. It builds on top of the Word2Vec

approach but adds learned word vectors for subword character N-grams. The word vectors can be then obtained by taking sum of the character N-grams. By taking subword information into account, the authors claimed that the model is able to form accurate word vectors for out of vocabulary words. The approach is particularly intriguing for languages with rich morphology, such as Finnish, where words often possess suffixes or can be fused together to form compound words.

In FastText model each word is presented as its character N-grams, as the model does not take the word or subword order into account these can be seen as a bag of character N-grams. Also, the whole word itself is included in the list of N-grams. Special characters `<` and `>` are added to the beginning and end of the word. For example, taken word `character`, the character N-grams, where $N=3$ are:

`<ch, cha, har, ara, rac, act, cte, ter, er>`

and the special case of the whole word `<character>` is added to the list of N-grams separately. From subword character N-grams like `er>` a meaning of something like acting can be learned and used when calculating word vector for example for `runner`.

According to the authors, the practical approach is to use character N-grams, where $N \in (3, 4, 5, 6)$ [5]. Each of the character N-gram and distinct word seen in vocabulary is associated with vector representation and the vectors are trained using Word2Vec model architecture. Either CBOW or skip-gram architecture can be used in FastText approach as well. According to authors, this rather simple setup boosted the performance of plain Word2Vec model significantly on various language modeling tasks [5].

To calculate a word vector for a word unseen in vocabulary, a sum of the word's character N-gram vectors is taken. Therefore, it is unlikely that there exist words that do not contain any character N-gram seen in training. If the whole word exists in the vocabulary, the vector learned for the whole word is used. To further get vector representation for a sentence or whole invoice containing multiple words, a similar method than for fusing Word2Vec vectors can be used.

A highly optimized C-code for training and using FastText vectors was made public by the authors¹. They also trained and published pretrained word vectors for 157 languages² including for example Finnish and English. The published word vectors are 300 dimensional and trained on the internet data from Common Crawl³. The pretrained word vectors can be further fine-tuned using the provided code and datasets more similar to the final usage scenario to increase performance and to introduce more new words and character N-grams to the model.

¹<https://github.com/facebookresearch/fastText/>

²<https://fasttext.cc/docs/en/crawl-vectors.html>

³<https://commoncrawl.org/>

The computational complexity of word vector computation and classification rises when the dimensionality of the word vectors grows. Therefore, it would be beneficial to distill the information of high dimensional word vectors to lower dimensionality. One possible way to reduce the embedding matrix dimensionality is to use principal component analysis (PCA). PCA is a widely used matrix decomposition technique closely related to SVD presented in Section 3.2.2. The published implementation of FastText contains word embedding matrix dimensionality reduction using PCA, where PCA is applied separately for matrices containing the whole word vectors and subword character N-grams.

3.3 Classification model

For classification, a widely recognized and competition-winning gradient-boosted ensemble model XGBoost [6] is used. The selection of the optimal classification model is not considered as part of this work. Therefore, the classification model from the previously used coding prediction system is utilized as it is implemented into the software as well.

The classification model takes as input the encoded invoice header data and feature vectors extracted from raw invoice content. The encoded header data and feature vectors are simply concatenated together to form one input vector to the model. The output of the model is a multi-class output, meaning probability distribution over possible values for one coding dimension, where then the most probable values can be selected. Next, the basis of XGBoost for multi-class classification is presented taking into account the requirements of this work.

The basic idea of the gradient boosting framework is to combine a set of weak learners to form one strong model. In the case of this work and XGBoost applications in general the weak learner is a binary decision tree. In general binary decision trees can be trained by finding the decision splits in the data that most improve the model, so have the highest gain. A decision node of the tree makes the split by setting the limit on one of the input features. Splits can be found for example using exhaustive search, where each possible split is evaluated and the split with maximum gain is selected.

3.3.1 Multi-class predictions with XGBoost

The intuitive idea of boosting is that each binary decision tree will learn some characteristics of data and the following trees can then correct the flaws of previous trees. To combine the predictions of multiple trees, boosting is used to sum up the results from many trees to get prediction \hat{y} .

$$\hat{y} = f_0 + \sum_{k=1}^K \eta f_k(\mathbf{x}), \quad f_k \in \mathcal{F}, \quad (3.8)$$

where \mathcal{F} is the set of K trees and \mathbf{x} is the input data vector.

Each tree f_k outputs a weight w from one of its leaf nodes that contributes to the final prediction. Therefore, the decision trees in XGBoost are regression trees, outputting continuous scores for a given input, rather than class labels in a normal decision tree.

The idea is that each new tree added to the model corrects the result of previous trees taking into account different aspects of the data. To control the contribution of a single tree into prediction a learning rate $\eta \in [0, 1]$ is used. Initial guess f_0 or *base score* is an arbitrary value to start the optimization. A better estimate of f_0 can be calculated from the distribution of training data depending on the learning task at hand.

For multi-class classification purposes set of trees \mathcal{F} is built for each class in a set of classes C . In the end, the predictions for each class are compared in a "one-versus-all" manner by proposing class with highest predicted value

$$\hat{y}_{multi-class} = \operatorname{argmax}_c \hat{y}_c, \quad c \in C. \quad (3.9)$$

In training phase targets y_i are one-hot-encoded. To compare the one-hot-encoded targets to model output, softmax function $\sigma(\hat{\mathbf{y}})_c = e^{\hat{y}_c} / \sum_{j=1}^C e^{\hat{y}_j}$ is used to turn the predicted values to a proper probability distribution. A minimalistic example of multi-class prediction with XGBoost is presented in Figure 3.6, with 3 classes and 2 trees forming the model.

3.3.2 Finding the optimal tree structure

Finding the optimal decision rules, the number of trees, and, the weights model is trained in a supervised manner for n data points. Following Chen et al. [6], the training objective in XGBoost is to minimize the regularized loss

$$\mathcal{L} = \sum_{i=0}^n l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \quad (3.10)$$

where $\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$ is a regularization term that penalizes for a high number of leaf nodes T in a tree and large absolute weights. γ and λ are regularization parameters and l is a differentiable loss function that measures the difference of prediction and target output. In the case of multi-class XGBoost, the classification error rate is used as loss $l_{\text{error}} = \text{wrongly classified} / \text{all samples}$.

To control the depth of trees, γ is used to regularize the amount of leaf nodes in one tree. Additionally, hyperparameter *max depth* can be set to limit the depth of individual trees to further reduce the splits made in on tree.

Trees are added to the model greedily so that the new tree f_t is the one that most improves

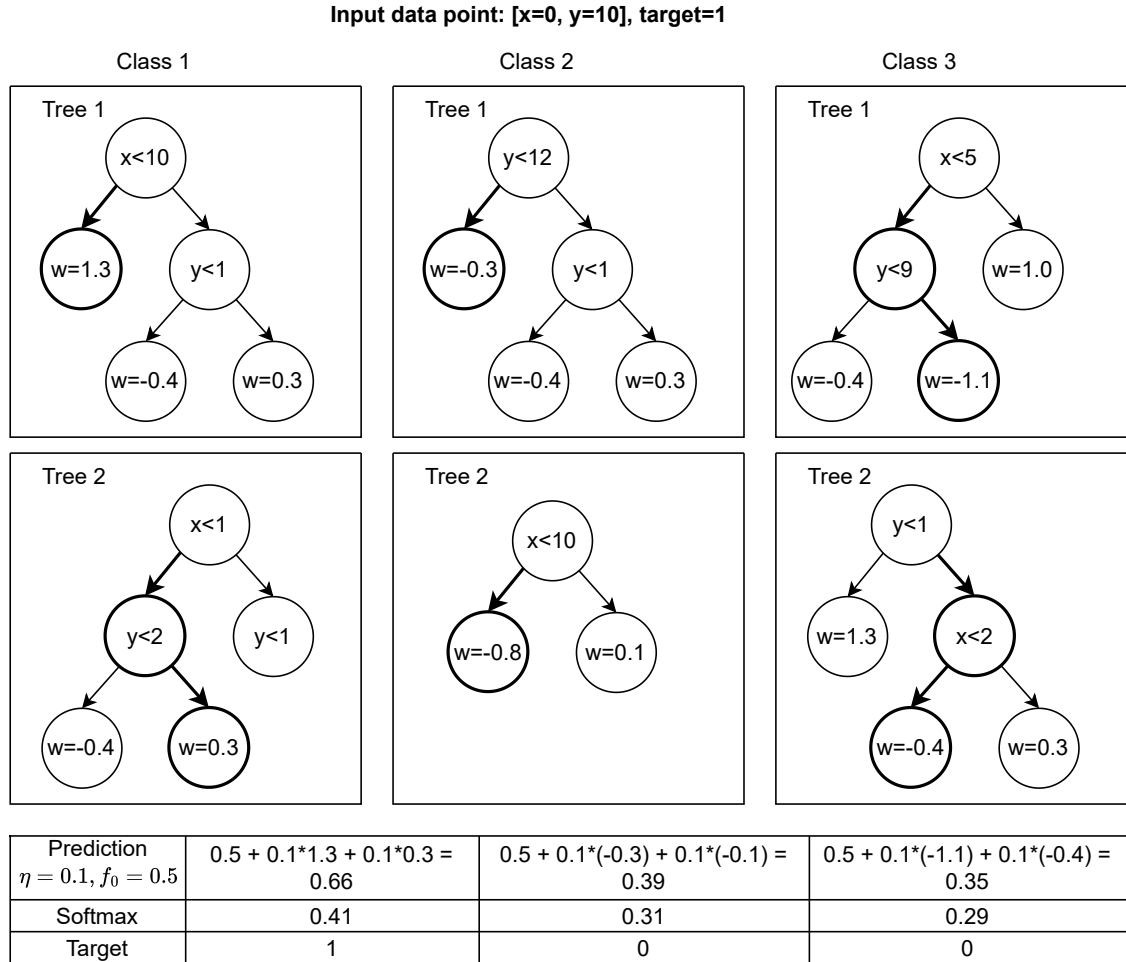


Figure 3.6. Single data point multi-class classification with tree ensemble using boosting. Each tree for class predicts a residual and the sum of residuals is calculated by using Equation 3.8. In the end, the softmax is applied to turn the predictions of each class into a proper probability distribution. Decisions made in each node are highlighted with this input data sample. Splits and tree structures have been found previously using separate training dataset.

the model. Newton's method for optimization can be used to approximate the optimal objective with second-order Taylor expansion, so that in step t

$$\mathcal{L}_t \simeq \sum_{i=0}^n \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_k), \quad (3.11)$$

where $g_i = \partial_{\hat{y}_{(t-1)}} l(y_i, \hat{y}^{t-1})$ is the first order derivative and $h_i = \partial_{\hat{y}_{(t-1)}}^2 l(y_i, \hat{y}^{t-1})$ is the second order derivative of the loss function respect to the current prediction.

From Equation 3.11 an approximation of optimal weight w_j^* and loss value $\tilde{\mathcal{L}}$ for making split in certain leaf node j holding data points I_j can be derived so that

$$w_j^* = - \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \quad (3.12)$$

and

$$\tilde{\mathcal{L}} = -\frac{1}{2} \sum_{j=0}^T \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \quad (3.13)$$

Now split in one decision node can be evaluated by calculating the gain from the approximated losses

$$\mathcal{G}_{split} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma, \quad (3.14)$$

where I, I_L, I_R are the data points of the root, left and right node respectively.

In the case of multi-class classification assuming the simple loss of l_{error} the loss needs to be formulated in differentiable form. One common option is to use logistic regression so that $l_{\text{CR}} = -\hat{y} \log(y) - (1 - \hat{y}) \log(1 - y)$. This is effectively the same as using l_{error} . By utilizing l_{CR} , the following formulas can be derived for first and second-order derivatives

$$\begin{aligned} g_i &= \hat{y}_i - y_i \\ h_i &= \hat{y}_i(1 - \hat{y}_i). \end{aligned}$$

So g_i describes the residual to the correct label and h_i the distance from the end of the prediction range.

In the training of XGBoost, a maximum value for the number of trees K is given as hyperparameter *number of trees*. The incremental nature of adding the trees in training however makes it possible to measure the additional benefit of adding new trees. Therefore, the training can be stopped when adding new trees does not improve the model further. This is a common technique in training ML models in general and is often referred as early stopping [32].

Although the optimal number of trees can be obtained in training time by utilizing early stopping, seeking optimal values for other hyperparameters in the XGBoost model is nontrivial. Educated guesses based on the dataset are valuable, but running experiments with different hyperparameter combinations are needed to end up in an optimal hyperparameter setting.

When taking the non-functional requirements, such as training and inference times into account in hyperparameter selection, it is evident that the more trees and splits the model makes the longer it takes to train and give predictions. Decreasing the values of regularization parameters reduces the number of splits in one tree, but more trees might be needed to reach the same functional performance. Therefore, hyperparameter optimization is always a tradeoff for satisfying functional and non-functional requirements.

3.3.3 Contribution of XGBoost

The new contribution in the XGBoost paper compared to earlier gradient boosting methods was the usage of regularized loss in Equation 3.10 and Newton's method in optimization. On top of that the authors proposed multiple tricks to make the computation more efficient and scalable.

To allow training on large data sets and even with parallel computation settings, an approximation of the greedy algorithm for split finding was presented. This algorithm first divides the data into weighted quantile buckets and the splits are then experimented on the quantile's aggregated statistics rather than individual data points. [6]. The h_i values are used as the weights so the data points with predictions far from the correct target have a higher probability of ending up to small quantile buckets, which makes the optimization easier.

Another key factor that makes the XGBoost so effective is that it has a built-in mechanism to cope with missing data points. In training phase when finding the splits, the missing values are tested for both child nodes, and the default direction for missing data is assigned to the direction of greater gain. [6]

To further speed up the calculations, a random subsampling of features was found to be useful for regularization on top of the traditional subsampling of data points. This allows the later trees in the process to find different features to split. [6]

Authors of XGBoost put a great effort into optimizing the computation and scaling it across multiple machines by introducing cache-aware access and blocks for out-of-core computation. They open-sourced the highly optimized software library⁴ that is nowadays widely used for different machine learning tasks.

3.4 Evaluation methods

Evaluation metrics are defined to investigate the performance of the proposed machine learning pipeline. The evaluation metrics in this work consist of metrics measuring the classification performance and metrics evaluating the non-functional requirements like execution time and memory consumption. This chapter defines the used evaluation metrics and introduces the stratified cross validation that is used to get the most realistic results from the data sample.

Ultimately the goal is to evaluate how different feature extraction techniques perform for coding dimension classification. The additional features from invoice content presumably help the classifier to make decisions. The techniques for comparing the features vectors to each other is not relevant, because the target is to provide information for classifier.

⁴<https://xgboost.readthedocs.io/en/stable/index.html>

Opting for vector comparison in the evaluation process carries the risk of overlooking intricate details within the features that the classifier can effectively leverage. This is particularly evident in the case of classifier like XGBoost, which dissects different feature values to enhance its decision boundaries. As a result, relying solely on vector comparisons for evaluation is likely to neglect the nuanced information embedded in the features. Consequently, the most viable and dependable approach to evaluate feature extraction methods is to compare the classifier's performance with varied features as input.

3.4.1 Evaluation metrics

For multi-class classification tasks, a common way to present the classifier predictions is to form a confusion matrix. A general confusion matrix for multiple classes is presented in Figure 3.3, where correct predictions are on the diagonal of the matrix.

		Predicted			
		Class 1	Class 2	...	Class k
Actual	Class 1	$n_{1,1}$	$n_{1,2}$...	$n_{1,k}$
	Class 2	$n_{2,1}$	$n_{2,2}$...	$n_{2,k}$
	⋮	⋮	⋮	⋮	⋮
	Class k	$n_{k,1}$	$n_{k,2}$...	$n_{k,k}$

Table 3.3. Multi-class confusion matrix for k classes. $n_{x,y} \in \mathbb{N}$ presents the count of data points where the model predicted class y and the actual class was x .

The simplest evaluation metric derived for multi-class classification is *accuracy*, which measures how many of the predictions were correct among all the predictions. From the confusion matrix accuracy can be calculated by taking sum of the diagonal elements and dividing that by the sum of all the elements in the confusion matrix. Formally,

$$accuracy = \frac{\text{correct}}{\text{all samples}} = \frac{\sum_{i=1}^k n_{i,i}}{\sum_{i=1}^k \sum_{j=1}^k n_{i,j}}. \quad (3.15)$$

While *accuracy* is a simple and easily interpretable metric, it fails to describe the class specific performance and gives more weight to classes that have many samples.

To address these drawbacks of *accuracy*, inspiration from binary classification evaluation metrics can be drawn. For example, the *F1* score has been found effective measure of performance for binary classification that takes the harmonic mean of *precision* and *recall* values [27]. For calculating binary classification metrics true positive tp , false positive fp , and false negative fn values for each class need to be calculated. From the confusion matrix for multiple classes, the class i tp_i is the value in the diagonal $n_{i,i}$. fn_i is the sum of values where actual class is i and prediction is not i . For fp_i the value is

the sum of elements where the prediction is i , but the actual value is something else.

From tp_i , fp_i and fn_i values, the metrics for class i can be calculated as

$$precision_i = \sum_{i=1}^k \frac{tp_i}{tp_i + fp_i} \quad (3.16a)$$

$$recall_i = \sum_{i=1}^k \frac{tp_i}{tp_i + fn_i} \quad (3.16b)$$

$$F1_i = 2 \frac{precision_i recall_i}{precision_i + recall_i}. \quad (3.16c)$$

To combine the calculated $F1_i$ scores for all classes, an average of the scores is taken [27]. This gives equal weight to each class. One possible way to balance the result to give more weight to bigger classes is to use a weighted average, where the weight of the class is given by the number of samples in class $|C_i|$. These metrics are typically called macro averaged and weighted average of the $F1$ scores and are formally presented as

$$F1_{macro} = \frac{\sum_{i=1}^k F1_i}{k} \quad (3.17a)$$

$$F1_{weighted} = \frac{\sum_{i=1}^k |C_i| F1_i}{k}. \quad (3.17b)$$

By measuring $F1_{macro}$ and $F1_{weighted}$ separately it can be determined how the classification successes overall and for individual classes. If $F1_{macro}$ score is lower than $F1_{weighted}$ it indicates that classification performs better on classes with many samples and worse on classes with fewer samples. It can be seen that the classification model therefore overfits the data of common classes and fails to generalize to uncommon classes.

Overfitting in general is a central problem in the design and implementation of machine learning systems [10, ch. 5.2]. The selection of appropriate classification algorithm and hyperparameters along with generalizable features are the methods to tackle the overfitting problem in the setup of this work. The main metric to control overfitting is the difference between train and test set performances. If the performance of the model equals on samples seen in training and unseen samples in the test set the model has learned the general rules of data and is not overfitted.

The non-functional metrics used in this work are the model training time and model memory footprint when it is saved to disk. Model training is computationally intensive and needs to be done on daily or weekly basis for models to learn the latest patterns of coding. Training the models requires computation, which naturally comes with cost. There-

fore, it is necessary that the training time remains feasible to be able to retrain often with negligible costs.

For experimenting the training time, it is necessary to use the same amount of computation capacity to reliably measure the training time as using a more powerful computer would presumably speed up the training of the ML pipeline.

Saving models requires memory capacity which also comes with cost. The amount of memory the model consumes indicates the response times in the inference phase as the model needs to be loaded to make predictions. Therefore, model size should not grow drastically.

3.4.2 Stratified cross validation

Cross validation is a method to get the most realistic performance for unseen data points and to ensure classification is not overfitting to particular datapoints. The idea of cross validation is to shuffle the data set and divide it into roughly equally sized groups or folds and then execute training and testing for each combination of folds. Usually, only one fold is used for testing, and the rest for training.

This way, evaluation metrics are calculated for each fold, which can be then analyzed statistically to form the best estimation of performance. For example, averaging of the metrics over folds gives an estimation of overall metric value, and also deviations of results can be analyzed to address how reliable the result is [10, ch. 5.3.1].

When randomly selecting the samples for each fold of cross validation it is unsure that the folds are representative samples of the whole data set. To address this problem the folds can be selected so that the class distribution for each class roughly follows the class distribution of the whole dataset. This approach is called *stratified cross validation* and it is illustrated in Figure 3.7 by Justin Lange⁵.

While dealing with imbalanced datasets, where the number of samples per class varies, stratified cross validation often gives more realistic results. However, to follow the class distribution of the original data set, the minimum amount of data points for a given class needs to be the same or greater than the number of folds in cross validation. If the numbers are equal, each sample of that class will be divided into different folds. In the scope of this work classes with fewer samples than the number of folds will be left out from the cross validation.

⁵<https://stats.stackexchange.com/q/452798>

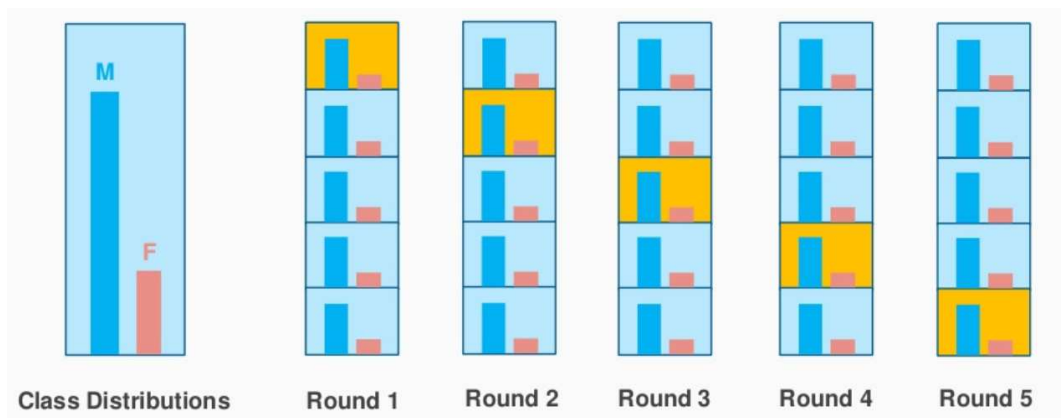


Figure 3.7. 5-fold stratified cross validation for 2 classes *M* and *F*. Each fold in cross validation shares the same distribution of classes as the original full data sample presented on the left. The fold highlighted in yellow is used as a test set for that round, other folds are used for training the model.

4. IMPLEMENTATION AND RESULTS

Experiments with real-life invoice data were conducted to assess the performance of various feature extraction techniques described in previous chapters. Compared to previous studies the data used has more variability and volume, which makes the results interesting also in academic viewpoint. This chapter gives some details about the used dataset, describes the used preprocessing techniques, used hyperparameters, and other implementation details of experiments.

The results are then presented in multiple phases. First, different feature extraction techniques are compared, secondly, the performance gains of using the invoice content for different dimensions are addressed. Finally, the results from experiments where also non-e-invoice were included in the dataset are shown. This setup is closer to the ultimate production setup, where all invoices regardless of the origin of the invoice need to get coding predictions.

4.1 Dataset Description

The invoice data is extracted from real customer databases. For the purposes of this work, it is not intentional to make the names of these companies public and in future steps, companies are just referred as Company A, Company B, etc. The different companies have significant differences in their invoice origins, processes and the language of the invoices making the dataset varied in multiple dimensions.

The coding dimensions included in datasets consist of account codes, cost center codes, internal orders, and cost allocations. Tax codes are not included, because over 90% or even 99% accuracy is reached with using header data only. The used coding dimensions bring varied aspects of coding to experiments. For example, cost center codes or related names may appear in invoice content as is, but account codes usually need the semantic meaning the bought items.

Table 4.1 describes the different characteristics of the used datasets. In the first part of the experiments, where the goal is to find an efficient feature extraction method, only e-invoices are used. Company D, despite lacking e-invoices, is included to substantiate the efficacy of the methods as they have captured the invoice data partially using OCR

or other capturing techniques. *E-invoice %* column in the table describes the share of e-invoices. Similarly, only invoices with one coding row are used for experiments and *1-CR %* column tells the share of invoices that have one coding row.

After taking only one coding row e-invoices for the latest 6 months an invoice set for each company is obtained containing invoice count described in column *Invoice count*. For each used dimension *Coverage* describes the share of invoices where the dimension value is populated in the dataset. *Unique values* tells the number of possible values for dimension and *Rare values* the number of these values that occur less than 5 times in the datasets. The invoices with rare dimension values are dropped from the dataset to make 5-fold stratified cross validation work.

The distribution of dimension values is very unbalanced meaning that for some values there are thousands of datapoints and for most only a few. The *std. of values* column in Table 4.1 presents the standard deviation of the invoices per dimension value to give an idea of the unbalanced distribution of labels. In Figure 4.1 distribution of invoices per dimension value is presented for Company A's account codes.

Company	E-invoice %	1-CR %	Invoice count	Dimension	Coverage (%)	Unique values	Rare values (<5)	std. of values
company A	48.0	82.8	35 911	Account Code	100.0	188	40	1537.6
				Cost Center Code	56.7	1334	386	966.0
company B	85.8	72.9	6 860	Account Code	100.0	211	32	154.8
				Cost Center Code	100.0	445	72	70.6
company C	69.3	80.3	7 701	Account Code	99.7	243	60	436.4
				Cost Center Code	92.5	967	334	112.9
company D	0.0	74.9	3 731	Account Code	100.0	126	60	137.9
				Cost Center Code	100.0	166	73	97.4
company E	96.4	72.0	16 073	Internal order	93.2	460	47	70.0
company F	96.4	88.9	47 003	Internal order	99.8	551	134	397.3
				Account Code	100.0	126	33	2019.6
company G	95.7	43.3	5 607	Cost allocation	100.0	284	126	399.5

Table 4.1. Characteristics of used datasets. Invoice count shows the invoice count after taking e-invoices with one coding row.

In Figure 4.1 the account codes of a single company are ordered in alphabetical order and form groups of codes that are assigned to many invoices. Two imaginary example account codes 034420 and 211120 are shown as the largest peak value of the group. The reason for groups in the graph is the hierarchy of the account codes described in Section 2.1.1, meaning that the granularity of the account codes increases towards the end of the actual account code number. Therefore, the similar and most used codes like account codes starting with 034 and 211 form spikes in the graph.

The datasets for different companies include invoices from May 2023 to November 2023. In previous studies, the invoice coding data was considered to be temporally dependent, meaning that it changes over time [21]. As the datasets used in this work are from a limited and relatively short time span, the data was considered to be temporally stable. In Figure

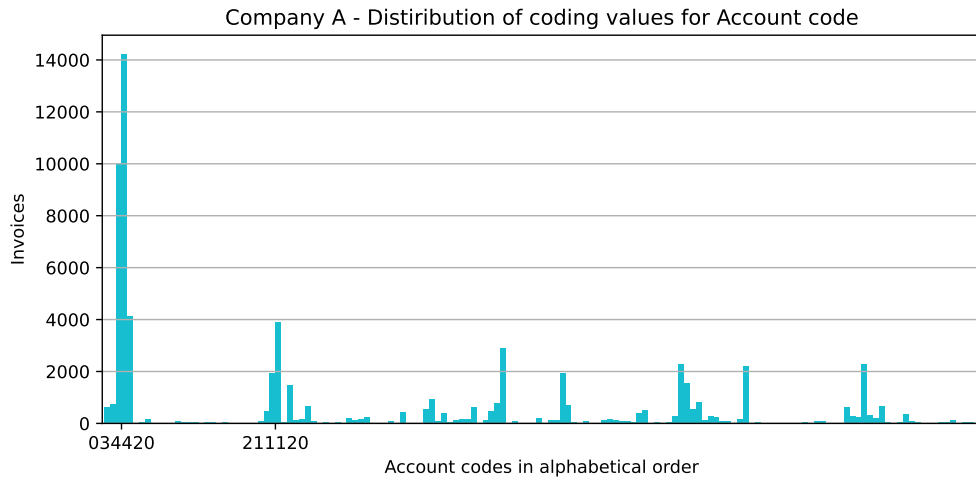


Figure 4.1. Example of coding value distribution. Coding values can be extremely unbalanced like here. The standard deviation of invoices per dimension value is 1537.6 in Table 4.1, which is one of the highest obtained numbers.

4.2 the used account and cost center values for each week are shown. New values that have not been seen previously are highlighted with crossed orange. For account codes, it seems that there are very few new values after about one month of data. Whereas for cost center codes it took about 2 months to reach the steady state. The amount of used values weekly stays stable, excluding the quiet summer weeks of July.

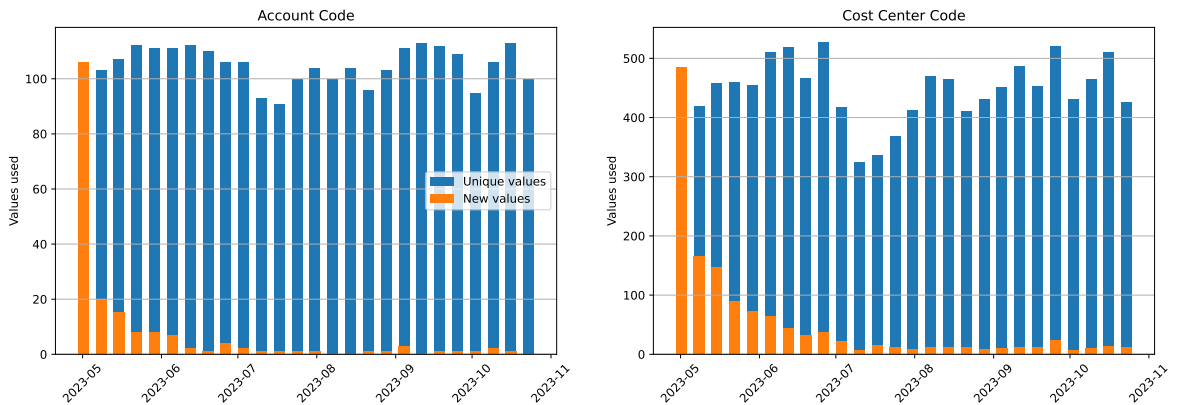


Figure 4.2. Used dimension values and share of values not seen previously in weekly batches for Company A. Usage of account and cost center codes seems to be temporally stable.

The invoice content data is extracted from the corresponding TEAPPS files. Table 4.2 shows some basic characteristics of the invoice content data. The tags in the table refer to a subset of tags in TEAPPS files where the text is extracted as explained in Section 3.2. The words refer to the number of words found from the valid tags. The exact method of determining what is a word is described in the following preprocessing Section 4.2.

From Table 4.2 it can be seen that there are usually around 40 to 60 words in one invoice.

	Tags per invoice		Words per invoice		Unique words	
	median	average	median	average	all	>1
Company A	18	22.9	48	65.4	157 502	73 425
Company B	17	24.0	34	54.6	42 589	18 513
Company C	17	22.4	42	56.6	48 283	22 958
Company D	3	3.8	10	11.2	11 357	1 127
Company E	29	62.3	73	149.9	77 488	50 077
Company F	27	26.7	56	61.1	109 773	41 018
Company G	20	21.4	39	45.1	32 372	11 025

Table 4.2. Statistics from invoice content data. Tags are only the selected subset of tags from TEAPPS files. Words are the words from these tags.

Overall the averages seem to be larger than the medians, which indicates that there are invoices with very large amounts of tags and words. One tag seems to contain around 2-3 words on average, but by looking at invoices it is evident that many tags contain only one word and others much more.

Invoices from Company D have clearly a lower number of tags and words in their invoice data because their invoice content data is extracted from PDF invoices. Also, Company E has clearly higher tag and word counts on their invoices. This is a consequence of company E having many invoice lines on their invoices, on average 20 lines.

Overall, it can be said that the datasets used in experiments are diverse in language, invoice origins, coding row counts, and used dimensions. Also, the classification problem is very unbalanced especially for account codes because of the hierarchal schema widely adopted by companies. Data is handled in future steps as temporally stable, so it is possible to use shuffling of data in training and evaluation. One invoice contains a relatively limited amount of words, where the feature extractor needs to calculate the additional features.

4.2 Preprocessing

The datasets include the header data which is in tabular format and the invoice content data which is one long string concatenated from the TEAPPS files. To convert the invoices to a format that can be processed by the proposed machine learning pipeline the two different types of data sources need to be preprocessed with different techniques. For header data, this means converting the data to numerical format, and for invoice content strings basic text preprocessing techniques are used before starting the feature extraction.

4.2.1 Invoice header preprocessing

Before feeding the invoice header data to the classifier, the data needs to be in numerical format. Some of the header values like net and tax sums are already in numerical format and can be left as is. However, categorical variables like supplier and organization identifiers need to be encoded into numbers.

The simplest way to convert labels to numbers is to assign a unique running number to each unique value of the variable. Understanding the inner decision making of XGBoost classifier explained in Section 3.3, reveals that the simple label encoding approach might not be the most efficient option. XGBoost creates the decision boundaries, where all datapoints with smaller or larger values will end up in different branches of a single tree. In a basic label encoder the order of the encoded labels does not contain any information about the actual labels, so datapoints will split randomly in decision nodes.

A better solution is to use a so-called target encoder where labels with the most datapoints are given the largest encoded values [17]. Therefore, decision nodes in XGBoost are able to split variables based on the frequency of the labels. In experiments conducted, categorical variables are encoded with a target encoder, and variables with already numerical values are left as is.

4.2.2 Preprocessing of invoice content text

Text preprocessing of the invoice content data aims to clean, unify, and reduce the noise in the data before the text is fed into the feature extractor [14]. First, the text is converted into lowercase characters and the characters are uniformed to follow the Unicode encoding. Then, the text is tokenized into words by splitting from whitespace and removing special characters like punctuation. A curated list of stop words are removed from the list of tokens containing words like *and*, *eur*, *day*. Also, tokens with under 3 characters are removed. Word normalization processes like stemming and lemmatization, where the idea is to consolidate different forms of the same word into the same token, are not used.

To cope with phrases meaning words occurring together forming a single meaning, there exist a couple of strategies. The statistical approach is to analyze the data and combine the pattern of tokens that occur often. These are typically called collocations in NLP. Another approach is to from N-grams introduced in Section 3.2.1 and let the feature extractor decide what phrases are important. This work takes the N-gram approach to express the entities formed by multiple tokens.

As seen in Section 3.2 the text is extracted by concatenating the values of XML tags. One tag usually contains relatively few words and neighboring tags contain information usually not related to the current tag. Some addresses might make an exception, where for exam-

ple road name and number are in different tags than the postal code and city. Therefore, the N-grams are formed inside a single tag so that tokens from neighboring tags do not end up in the same N-gram. This requires the usage of the structural information of the data, so the preprocessing does not rely fully on unstructured data sources. Also, only a curated subset of the tags are used to form the strings. For example, all tags containing amounts and unique identifiers are excluded along with all invoice routing metadata.

4.3 Implementation details

This chapter describes the used hyperparameters for feature extraction and classification. After reading this chapter the reader should be able to understand the methods and notation used for tables presenting the results in the following chapters.

The experiments were run using cloud computing of Amazon Web Services (AWS) utilizing the SageMaker¹ environment. The used computing instance was selected to be `m1.m5d.xlarge` with 4 CPU cores and 16 GB of memory. Python 3.8 was used to implement the machine learning pipeline utilizing libraries like Pandas², scikit-learn³, NumPy⁴ and XGBoost⁵.

The results were obtained using 5-fold stratified cross validation introduced in Section 3.4 that allowed to observe the stability of the results and get the best estimate of metrics by taking an average over the folds. The baseline results are always executed without any features from the invoice content using only the header features. The features used from the header vary between companies because of the differences in their invoice process. The baseline result reflects the performance of the currently used coding prediction system described in Section 2.3.

For Bag of Words (BoW), introduced in Section 3.2.1, the distinct words were calculated from the token N-grams extracted in the preprocessing phase. The different values for N in N-grams are tested ranging from 1, meaning only the individual tokens to 3. Notation 1-3 grams means that all 1, 2, and 3 grams are fed into BoW as distinct words. When calculating the BoW matrix N-grams with less than 50 occurrences are not considered. Also, N-grams that appear in over half of the documents are dropped. Limits are set to limit the number of distinct N-grams in the BoW matrix for more efficient computation.

From the BoW matrix, duplicate N-grams are searched and only one of the duplicates are left in the final BoW representation. This reduces the BoW dimensionality further without losing any information. After duplicate removal BoW matrix is converted to a binary matrix

¹<https://aws.amazon.com/sagemaker/>

²<https://pandas.pydata.org/docs/index.html>

³<https://scikit-learn.org/stable/>

⁴<https://numpy.org/>

⁵<https://xgboost.readthedocs.io/en/stable/index.html>

by assigning 1 to each cell where the value is 1 or greater.

After dropping the frequent, rare, and duplicate N-grams the number of distinct N-grams is still in the range of many hundreds to thousands and needs to be further reduced to use as features. Two methods to use BoW as a feature extractor are tested. The most frequent method takes the selected amount of words that are the most frequent among the invoice dataset. Most correlative methods calculate Theil's U correlation from Equation 3.2 and take the N-grams that have the highest correlation with the target dimension values.

The binary BoW matrix is used as a base matrix for the LSA method. Furthermore, the amount of components in SVD dimensionality reduction done by LSA needs to be selected and is shown in the brackets like LSA(20) in tables presenting the results. Another option is to not use the binary BoW matrix but to use the TF-IDF weights as in Equation 3.1. The SVD in the LSA computation is very fast, taking only a few seconds on the used datasets.

The Word2Vec and FastText feature extraction methods are tested using the Gensim⁶ library. The feature vectors are calculated from the same tokens used for other methods. The main hyperparameters tested were the effect on the feature vector dimensionality, again noted in brackets, and the model architectures CBOW and skipgram. For pretrained FastText experiments, the model was used as is, or by reducing the dimensionality as described in Section 3.2.4. This kind of reduction is noted as *red.* in result tables, whereas *PCA red.* refers to the technique where the resulting feature vectors from the pretrained or finetuned model were reduced by PCA.

Word2Vec and FastText were trained using the window size of 5 over the whole dataset for 5 to 20 epochs. The results only show the best obtained models. The training times for models ranging from 1 minute for a simple Word2Vec CBOW model with low feature vector dimensionality to 20 minutes of FastText skipgram architecture with high 100-dimensional feature vectors.

To calculate the feature vectors for the document the feature vectors produced by Word2Vec or FastText needed to be combined. As seen in Section 3.2.3 the vectors can be combined either by averaging the features over the words or by weighting the average with respective TF-IDF weights. Averaging was the default method used in the experiments and usage of TF-IDF weights is denoted with *tf-idf* in Word2Vec experiments.

The evaluation pipeline was set so that the feature extraction needs to be performed for the whole dataset first and then the cross validation is done. There exists a potential risk that features carry some information about training to the test set resulting in overfitting in classification. However, some of the results were ensured to hold even when the feature extraction was done as part of the cross validation. This limitation also makes it unreliable

⁶<https://radimrehurek.com/gensim/>

to measure the time used for feature extraction so only time to train the classifier is shown in the results.

Description	Value
Learning rate η	0.3
Base score f_0	0.5
Gamma γ	0
Lambda λ	1
Number of trees	100
Max depth	6
Eval. metric	merror

Table 4.3. XGBoost parameters used for classification. All experiments used the same setup for XGBoost.

For classification, the default hyperparameters of the XGBoost library are utilized and no hyperparameter optimization is done. All feature extraction methods leverage the same hyperparameters to ensure fairness in comparisons. The used main hyperparameters are tabulated to Table 4.3.

4.4 Comparison of feature extraction techniques

This chapter compares the different feature extraction techniques for datasets. Table 4.4 starts by presenting the results for Company A's account code. The average values for metrics are presented over 5-fold stratified cross validation and the difference to baseline results is shown as an absolute value.

Method	Feature dim.	Accuracy	$F1_{macro}$ score	$F1_{wgt.}$ score	Training time (sec)	Model size (MB)
Baseline	22	65.8	57.0	65.6	132	18.8
BoW(100) most frequent	122	+6.1	+8.2	+6.0	+286	-1.1
BoW(100) most correlative	122	+4.2	+5.4	+4.1	+279	-0.6
LSA(20) 1-grams	42	+5.9	+7.7	+5.8	+203	-2.8
LSA(20) (1-3)-grams	42	<u>+7.5</u>	<u>+9.0</u>	<u>+7.4</u>	+200	-2.8
LSA(20) tf-idf weights	42	+5.5	+6.6	+5.4	+199	-2.4
Word2Vec (30) tf-idf weights	52	+0.8	+1.6	+0.8	+151	-0.6
FastText (30) skipgram	52	-2.6	-5.0	-2.7	+244	-0.6
Word2Vec (100) skipgram	122	+7.4	+5.8	+7.2	+1038	-3.3
FastText (100)	122	+2.9	+0.4	+2.6	+854	<u>-3.5</u>

Table 4.4. Classification results of different feature extraction methods for Company A's account code. Best results underlined. Note that training time is only the classifier's fit time, time for feature extraction is not measured.

In Table 4.4 LSA utilizing the 1-3 grams and forming 20 semantic topics performs the best

among all the functional metrics with one of the smallest increases in training time. The classification model size seems to stay relatively stable across feature extraction methods which is expected taking into account that the models use the same hyperparameters and therefore build about the same number of trees and nodes.

It is evident that for this dataset taking bigrams and trigrams along with unigrams makes more expressive features as LSA with only unigrams scores lower. Word2Vec with skip-gram architecture comes close to the best method, but utilizes more feature dimensions and is therefore much slower to train. It is notable that in accuracy and weighted F1 score the Word2Vec skipgram with 100 dimensional features comes very close to best method, but for macro averaged F1 score it falls back more. This means that utilizing more features probably causes the classifier to overfit to more frequent classes and fail to predict the less frequent classes.

Figure 4.3 shows the classification performance with the LSA feature extractor for different number of LSA components or topics. In the first datapoint, where LSA feature dimensionality is zero, the figure displays the performance of the baseline without any invoice content features. The rest of the datapoints used 1-3 grams as a base for LSA and extracted different number of features inputted to the classification model.

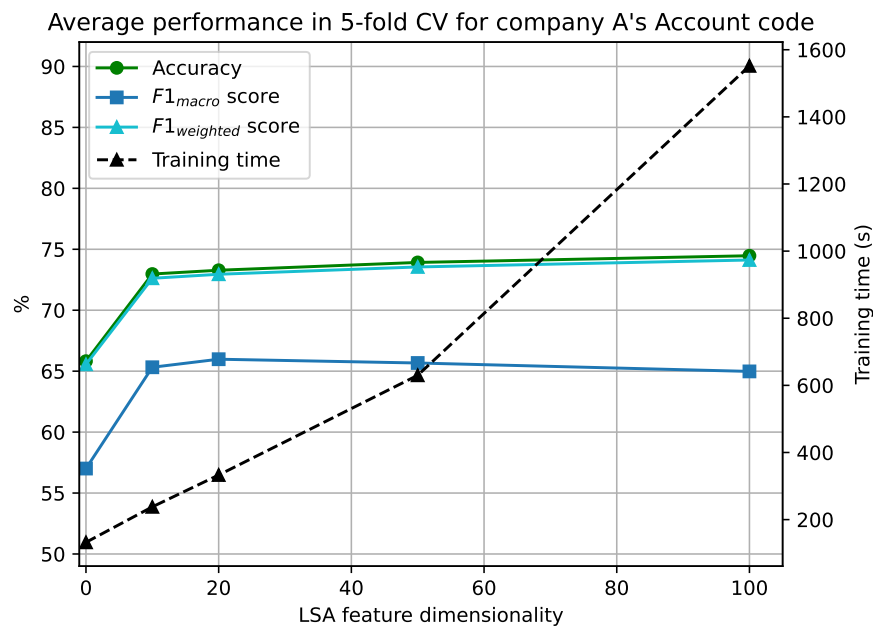


Figure 4.3. The effect of LSA components extracted to classification performance for company A's account codes. Zero LSA features means invoice content is not utilized.

Including even low dimensional LSA features gives a significant performance boost as illustrated by Figure 4.3. Increasing the dimensionality of LSA features only slightly improves the accuracy and $F1_{weighted}$ score, but using more than 20 components decreases the values of $F1_{macro}$ score, meaning worse performance for classes with few samples.

The training time rises linearly or even exponentially when adding more components. By the results of Figure 4.3 the best dimensionality of LSA features was determined to be 20.

Table 4.5 displays more comparisons of feature extraction methods for different datasets and dimensions. For companies E and F pretrained FastText models are experimented as feature extractors using the pretrained vectors as is, by finetuning the model and reducing the dimensionality of feature space. As previously the baseline result does not utilize any features from the invoice content. The best results are highlighted for each company and results are presented as absolute differences to the baseline result.

	Dim.	Method	Feature dim.	Accuracy	$F1_{macro}$ score	$F1_{wgt.}$ score	Training time (sec)	Model size (MB)
company A	Cost center code	Baseline	22	66.7	21.2	63.7	345	63.8
		BoW, 100 most correlative	122	-0.3	-3.5	-1.3	+604	-27.7
		LSA(20), 1-3 grams	42	+5.0	+9.8	+6.0	+739	-9.2
		FastText(30) skipgram	52	-3.7	-4.4	-4.1	+560	-17.3
company E	Internal order	Baseline	11	43.9	28.0	43.4	161	60.5
		BoW, 100 most correlative	111	+21.3	+23.6	+21.1	+461	-1.9
		LSA(20), 1-3 grams	31	+13.5	+11.5	+13.3	+411	-3.5
		Pretrained FastText(300)	311	+17.5	+18.7	+17.3	+2402	-31.2
		Pretrained FastText, red. (100)	111	+15.3	+15.7	+15.1	+875	-28.5
		Finetuned FastText, PCA(20) red.	31	+11.6	+10.1	+11.6	+207	-21.7
company F	Internal order	Baseline	10	53.4	34.3	50.2	315	65.1
		BoW, 100 most correlative	110	+17.8	+13.8	+19.9	+1424	+0.6
		LSA(20), 1-3 grams	30	+20.6	+17.6	+22.7	+1110	+0.1
		Pretrained FastText(300)	310	+19.2	+18.5	+21.8	+10352	-28.7
		Finetuned FastText(300)	310	+20.1	+18.4	+22.8	+9685	-26.6
		Finetuned FastText, red.(100)	110	+18.5	+11.8	+21.0	+3561	-25.1
	Account Code	Baseline	10	85.8	63.0	85.5	54	12.8
		BoW, 100 most correlative	110	+2.5	+2.3	+2.7	+254	-0.2
		LSA(20), 1-3 grams	30	+3.5	+2.4	+3.6	+202	+0.1
		Finetuned FastText, red.(100)	110	-6.6	-4.6	-7.2	+5951	+46.3
company G	Cost allocation	Baseline	8	75.1	38.0	74.0	7	9.7
		BoW, 100 most frequent	108	+2.7	+3.2	+3.0	+27	-1.1
		BoW, 100 most correlative	108	+3.6	+4.7	+4.0	+26	-1.2
		LSA(20), 1-3 grams	28	+2.5	+2.5	+2.7	+10	-2.7

Table 4.5. Method comparisons for more companies and dimensions.

By analyzing the results of Table 4.5 it can be said that LSA with 20 topics and using 1-3 grams as a word presentation works well for company A's cost center codes and for company F's both dimensions. It also improves the coding performance for company E's internal order and for company G's cost allocation dimension, although BoW features with 100 most correlative words to coding give even greater improvement. In these two cases, there seem to be strong links between certain words and coding dimension values that can be found by correlation analysis. Other methods are probably somehow presenting these words in the extracted features, but fail to capture some of the words or correlations.

The pretrained FastText experiments for companies E and F reveal that finetuning the model with the invoice content data does not make the features significantly more ex-

pressive. It can also be seen that reducing the dimensionality of the pretrained model decreases the performance on functional metrics, but reduces the training times as expected.

Using 300-dimensional pretrained or finetuned FastText features for company F's internal orders performs almost on par with LSA feature extraction. Actually, it seems that FastText features are able to capture more expressive features also from unfrequent classes as the $F1_{macro}$ score is higher. The LSA, however, uses only 20 dimensional features and therefore the computation time needed to train the classifier is 10 times less and also the feature extraction part is simpler and more robust.

There are some experiments that perform worse on functional metrics than the baseline like Company A's cost center code prediction with FastText(30) skipgram features and finetuned and reduced FastText features for company F's account codes. The feature extractors in these cases are failing to make features that present the invoice content in a meaningful way that could be understood by the classification model. For the latter case, the model's size and training time are large which may indicate that the model finds patterns from the content features that have no causal effect on the coding, so it overfits.

4.5 Performance gains from using invoice content data

The purpose of this chapter is to evaluate how much the use of invoice content data improves the coding accuracy for different dimensions. As seen from the earlier results, LSA feature extraction gives solid improvement to coding across almost all datasets and dimensions. This chapter provides more comparisons of using LSA features compared to baseline classification performance and digs deeper into the reasons why some companies and dimensions benefit more from using features from invoice content.

Figure 4.4 presents comparisons of classification accuracy results of using the baseline classifier with only invoice header data compared to the additional use of LSA feature extraction. All LSA results are computed by using 20 LSA topics and 1-3 grams as underlying BoW representation. Results for account codes and cost center codes are shown side by side for 4 companies. The results are obtained by using 5-fold stratified cross validation and taking the mean of the fold accuracies. There was no significant variation in the results between cross validation folds.

From Figure 4.4 it can be interpreted that only company A benefits significantly from using LSA features to account code predictions. Company A also has the lowest classification accuracy for the baseline method. By looking at the data, it seems like the supplier is a less significant explaining factor for account code for company A than for other companies. There are more suppliers whose invoices are assigned to multiple account codes. This means that they need to look at other things like item descriptions when assigning account

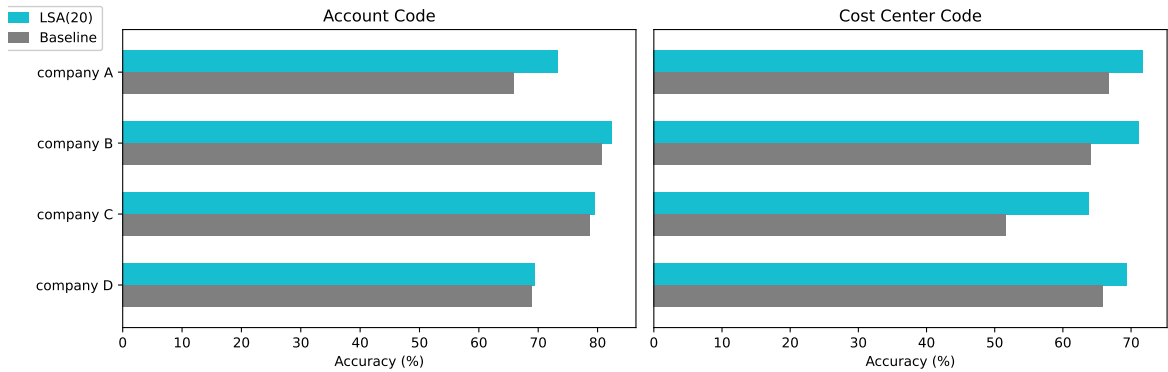


Figure 4.4. LSA (20) with 1-3 grams as word representation and baseline results for account code and cost center code classification accuracy. Cost center codes seem to benefit more from utilizing the features from the invoice content.

codes. Judging from the improvement the LSA features give, it may be that the classifier has learned to use some semantic meaning from the item descriptions by looking at the LSA features.

For companies B and C, account codes benefit slightly from using LSA invoice content features on top of invoice header data. However, they have relatively high baseline accuracy, and the receiving organization and supplier of the invoice seem to have a strong correlation to account code. Therefore, the additional features from invoice content do not give the classifier new information that would improve the accuracy.

For company D, with no E-invoice data, but only partially captured text from PDF invoices, LSA features are not beneficial for account code predictions. This is expected because the partial PDF data does not basically contain any information about the invoice lines, but only customer references on top of information that is already in the invoice header.

The classification accuracy for cost center codes significantly improves across all companies by utilizing the invoice content data with LSA feature extraction based on the right side of Figure 4.4. This means that LSA features from invoice content have been able to capture some of the actual cost center codes, related names, or addresses. Similar results for other management accounting dimensions like internal order can be obtained from the Table 4.5 in the earlier chapter. There, company F is measured to get over 20%pt. improvement for internal order code classification accuracy whereas accuracy for account code predictions improves only by 3.5%pt.

Overall, the performance gain by utilizing the invoice content data looks to be greater for management accounting dimensions like cost center and internal order. As seen in the Section 2.1.1 there usually are some words or phrases hinting at the correct dimension value like names, delivery addresses, or the codes itself that the feature extraction is able to deliver to the classifier. For account code predictions only minor improvements to classification performance are measured because of stronger correlations of account

codes to header data.

It can be also speculated that the LSA feature extraction is not complex enough to extract actual meaning from invoice line items that are described in natural language. Like if invoice line items are rakes, the most probable account code is an account for garden supplies. However, Word2Vec and FastText which are more complex and expressive techniques still fail to improve the account code predictions as seen from Tables 4.4 and 4.5.

4.5.1 Results for account code hierarchy

It is not known what is the maximal accuracy that account code predictions can reach in this single-label setting. As stated in Section 2.1.1 account codes usually are constructed by using hierarchy, where the general class is presented by the first number and classification granularity increases towards the end of the code. It can be that users code the invoices differently by using the neighboring accounts or more general accounts in the hierarchy. The used evaluation metrics do not favor predictions that are close together in the hierarchy but only calculate the correct and incorrect coding values.

Table 4.6 demonstrates the classification accuracy for different hierarchy levels of account codes. In the case of company A, all account codes are 6 digits. The accuracies in the table are calculated so that the level indicates the first n digits used from the account code to calculate the matches between predictions and true labels. For example, if the predicted account code is 410340 and the true account code is 410660 the codes match in levels 1, 2, and 3.

The accuracy is calculated using a separate test set, so the final level 6 accuracies that compare the full account codes do not exactly match with the cross validation accuracies in earlier tables. Results are shown separately for the baseline that does not use the invoice content data and LSA feature extraction with 20 topics and 1-3 grams as underlying BoW representation.

Level	Baseline	LSA(20)
1	85.0	88.7
2	78.2	83.1
3	77.9	83.1
4	72.0	79.4
5	65.1	73.5
6	65.0	73.5

Table 4.6. Accuracies for Company A's account codes using the hierarchy of 6-digit account codes. Level 2 for example shows the accuracy calculated by comparing only the first 2 digits of predicted and true account codes.

By investigating the results in Table 4.6 it seems that the accuracy does not change between levels 2 and 3 or between 5 and 6. Therefore, it seems that company A's account codes form a hierarchy that uses format X-XX-X-XX. This 4-level hierarchy has 7 unique values on level 1, 26 on level 2, 10 on level 3, and 44 on last level.

As expected the accuracy of the classification decreases when lower levels are tried to be predicted. Still, the level 1 and 2 accuracies are surprisingly low, so even the higher levels in the hierarchy are sometimes misclassified. This means that increasing accuracy over 90% will be almost impossible without deeper knowledge of the company's accounting practices. However, the low accuracies of high-level accounts show that there is definitely room to improve. The assumption that the human coded labels from data differ in lower levels in account hierarchy or utilizes sometimes the different levels of hierarchy seems to therefore be wrong.

A small interpretability study was conducted by forming a multi-class confusion matrix as presented in Table 3.3. Separate confusion matrices were formed for baseline and LSA feature extraction. The same data and setup of Company A's account codes as previously in this section are used to calculate the confusion matrices. The actual matrices are not shown because unmasked account codes are plotted to axis labels to make sense of the results. As there are hundreds of unique account codes used, the confusion matrices are also impractically large to present in this work.

By comparing the confusion matrices it looks like the LSA feature extraction allows the classifier to more accurately predict among the neighboring lowest-level accounts describing almost the same expenses. Otherwise, the misclassifications made by both models seem to be similar.

Both models almost equally fail in reliably determining even the highest level account on invoices where the correct account code describes project costs, freights, or costs for tools. It seems that these invoices coded to these accounts are partially from the same suppliers and might therefore be that some costs of invoices describing the same things are assigned to a project and some not. Thus models have not learned to fully distinguish whether expense is related to a project or not.

4.6 Including non-E-invoices to data

To prove that the added features from invoice content give value in production setting, the proposed ML pipeline will need to handle invoices that are not delivered as e-invoice. Some invoices might not have invoice content defined at all like invoices that are manually entered into the system and some have partial invoice content data like invoices captured manually.

Therefore, the proposed ML pipeline needs to be able to give predictions even if the

invoice content data is missing or incomplete. In the real production setting this is the reality that the proposed ML pipeline will face. Additionally, it is preferred that there is only one ML pipeline that works for all invoices in production setting to reduce the complexity.

The experiments in this section use a different set of invoices than the earlier parts because invoices with partial and missing invoice content are included. Previously only e-invoices that certainly have content data defined were used. The datasets used in this section are described in Table 4.7. Only part of the companies were selected based on their invoice origin characteristics. *E-invoice %* in the table shows the share of invoices that are e-invoices and have full invoice content data defined. *No content %* displays the share of invoices that do not have content data at all. The rest of the invoices come from various origins, but have some content data, containing presumably only parts of the text from the invoice.

	Invoice count	E-invoice %	No content %
Company A	67 614	46.0	32.1
Company B	8 367	83.4	4.9
Company C	12 220	63.9	12.0

Table 4.7. *Datasets used for this section. More invoices are included by taking invoices with partial and no content data to verify the usability of the proposed ML pipeline in real-life use cases.*

For each dataset in this section described in Table 4.7 baseline results were calculated by taking only the invoice header data. 2 different approaches for using the invoice content features were tested. In the first one invoice content data is used only from the e-invoices and features from invoice content are marked as empty for all non-e-invoices. The second approach takes the invoice content data from all invoices that have it, meaning e-invoices and those invoices that have partial content data. Invoices with no content only have the header data so for those, the content features are filled with empty values.

The 2 different approaches that use exactly the same invoices, but have differences in the features derived from the invoice content are marked as *Features from E-invoice, empty to others* and *Features for all invoices with content data* in Table 4.8 respectively. The assumption is that using only e-invoices gives improvement over baseline and using content from all invoices gives even greater improvement as there are some data available for more invoices.

The results in Table 4.8 are calculated similarly as in previous sections using the average of results in 5-fold stratified cross validation and using LSA with 20 topics and 1-3 grams as the underlying BoW representation. The results are shown as absolute differences compared to the baseline of that experiment. The column *Empty features (%)* describes the share of invoice content features that have empty values. Note that these slightly

differ from shares in Table 4.7 and between dimensions, because the evaluation pipeline drops invoices with rarely used coding dimension values.

	Dim.	Method	Empty features (%)	Feature dim.	Accuracy	$F1_{macro}$ score	$F1_{wgt.}$ score	Training time (sec)	Model size (MB)
company A	Account Code	No features from content	-	22	61.6	46.1	61.0	336	27.6
		Features for E-invoices LSA(20), empty to others	54.4	42	+7.2	+15.2	+7.5	+500	-2.1
		Features for all invoices with content data, LSA(20)	31.7	42	+8.8	+18.7	+9.2	+738	-3.2
company B	Account Code	No features from content	-	16	81.5	71.1	80.7	31	19.4
		LSA(20) with 1-3 grams, features for E-invoices	16.6	36	+1.1	+0.6	+1.3	+95	-1.7
	Cost Center Code	No features from content	-	16	66.0	50.9	64.7	56	35
		LSA(20) with 1-3 grams, features for E-invoices	15.8	36	+4.6	+6.0	+4.4	+150	-3.9
company C	Account Code	No features from content	-	16	78.8	65.5	77.8	39	19.9
		Features for E-invoices LSA(20), empty to others	36.0	36	+0.9	+1.3	+1.2	+106	-0.8
		Features for all invoices with content data, LSA(20)	11.8	36	+0.4	+0.1	+0.4	+140	-0.8
	Cost center code	No features from content	-	16	54.2	41.7	52.4	106	50.6
		Features for E-invoices LSA(20), empty to others	36.5	36	+6.4	+7.0	+6.5	+246	-4.4
		Features for all invoices with content data, LSA(20)	12.0	36	+6.7	+7.5	+6.7	+311	-4.6

Table 4.8. Including invoices without invoice content data. For companies A and C experiments without invoice content, invoice content from E-invoice only and content from all available invoices are presented. The proposed method increases the performance even if most invoices have no content data available.

The results in Table 4.8 show that including invoices with partial and no content can be done without losing performance gains compared to baseline results. As seen in Section 3.3 XGBoost has special handling for empty values and therefore the same model can handle invoices with and without invoice content.

Comparing the results of Table 4.8 to the same companies in Figure 4.4 shows that including the non-e-invoices into the dataset slightly decreases the performance gains. This is expected as invoices with limited data should be more difficult to classify. Still, the results are in line with the two illustrations.

For company A and C results with features from only e-invoices and features from all invoices are shown. Company A's account code predictions seem to benefit from including non-e-invoices to feature extraction as the accuracy increases by 1.5 percentage points. This means that the limited data from non-e-invoices still contains information that is utilized by the classification model when making predictions. The words in the invoice content are therefore at least partly the same.

For company C, including the non-e-invoices to feature extraction did not bring additional performance gain and even decreased the classification performance for account code. It may be that the partial data in the non-e-invoices is more limited than for company A. Also,

the baseline classification performance is much better for company C than for A, which affects the gain observed when including the invoice content features. Prediction of cost center codes for company C slightly benefits from including data from non-e-invoices, so the partial data seems to contain some cost center related information.

Generally, the proposed feature extraction methodology with the selected classifier XG-Boost seems to be able to handle partial and missing invoice content well. That said, it is important to have enough invoices with content data in the dataset as the feature extraction is only trained with those invoices.

4.7 Methodology critique

Looking back to the design decision of the implementation and the results obtained, there are a few things that can be criticized. For further development and research on this topic, it is important to be able to understand the limitations and possible hazards caused by these choices.

The first critique is the fact the cross validation used features that were first calculated for the whole dataset. The feature engineering, encoding of the invoice header, and the feature extraction of the invoice content features were done first for the whole dataset, rather than a separate training and test set for each cross validation fold.

It might be that this caused the features to be more expressive as the invoices from the test set are available in the training phase of the extractor. Therefore, there can be some information leakage from the test set that gives performance improvement in classification that cannot be obtained for unseen invoices. However, it was verified for some of the experiments that doing the training of the extractor inside the cross validation did not affect the average classification performance.

The second critique is to hyperparameter selection of the BoW model that acted as the baseline word extractor also for LSA feature extraction. Specifically, the selection of the pruning parameters of the vocabulary meaning the words that are dropped due to the high and low frequencies, and the selection of stopwords were questionable. These parameters were manually tuned by trial and error by looking at the size and frequencies of the resulting vocabulary. The main tuning job was done for only one dataset from one company and other companies just adopted the same parameters although the characteristics of the data were different. The same goes for the selection of the LSA component count that was only properly assessed for one of the datasets.

There is uncertainty in the removal of duplicated words and N-grams that was done to reduce the size of the vocabulary. In the production setting, it might be feasible to store a mapping of the duplicated words to the ones that remained in the training data. However, this introduces more complexity to the system and better approaches for finding

collocations can be examined.

When examining the invoice content data used for feature extraction, it appears to include irrelevant information such as arbitrary sums or codes. This prompts considerations regarding the optimization of valid tag selection from the TEAPPS files or employing additional preprocessing steps to enhance data cleanliness. One potential avenue for exploration involves manually curating data from selected invoices and assessing the impact on classification performance.

While experimenting with Word2Vec and FastText models the same corpus was first used in the training of the model and then features were extracted from the same corpus. This raises concerns regarding the possibility of overfitting the feature vectors to the data, potentially resulting in new invoices failing to generalize effectively. Furthermore, the limited scope of the training corpora, coupled with the sole reliance on training loss for measuring the learning process, could lead to underfitted feature extraction models.

The last critique is about using the same hyperparameters in the XGBoost classification model for every company and experiment. Experiments where features were added from the invoice content used at least twice as many features as the baseline experiments. Therefore, the optimal XGBoost hyperparameters might be different. When using more features the model is more likely to overfit because there are more possible splits to make in the dataset. Therefore, a lower learning rate and carefully setting the regularization parameters should be considered. Training more expressive models with more and deeper trees might lead to better performance in experiments with more features, although overfitting is more likely.

Overall, there are several factors to consider when interpreting the results and design decisions. Some of the decisions might have led to overfitting causing the result obtained for unseen invoices will be poorer than shown. Other decisions are related to assumptions that need to be made in the implementation phase like vocabulary pruning and collocation finding.

5. CONCLUSIONS

This work introduced a machine learning method to give coding predictions in a multi-class classification setting utilizing invoice content data as a newly enabled unstructured data source. Compared to the previous implementation of machine learning based coding prediction system that only leverages the invoice header data, the new method includes a feature extraction phase, where the text from the invoice is transformed into features of the classifier.

Based on previous academic efforts on automating coding prediction, bag of words (BoW) and Word2Vec were identified as potential methods to do the feature extraction. In addition, latent semantic analysis (LSA) and FastText were considered as extensions to previous methods. Different variants of these methods were tested as feature extractors in the experimental setup and compared against the baseline result utilizing only the invoice header data.

The experiments conducted used datasets extracted from 7 different companies that have various invoice processes, origins, and languages in the invoices. Compared to previous works the scale and variability of the data are considerably larger setting unique requirements for the robustness and scalability of the proposed machine learning pipeline.

Results from the experiments show that utilizing the invoice content data yields an increase in coding prediction performance. Almost all the used methods give some improvement, but across all the companies and coding dimensions tested, LSA feature extraction was shown to give solid and robust performance enhancement. Against the hypothesis at the beginning of the work, management coding dimensions like cost center and internal order code benefit more from utilizing invoice content features than account codes. This means that the feature extraction can retrieve references like names, codes, and addresses from the invoices rather than the semantic meaning of items described.

The coding prediction accuracy improvement over the baseline was measured to be around 1 to 8 percentage points for account codes and 2 to even 21 percentage points for management dimensions. The additional accuracy comes with the cost of increased training times of the classification models which can be 3 to 10 times longer. The measured training times are still feasible and also the sizes of the classification models stay low so the inference speeds are not affected.

In addition, the proposed machine learning pipeline with LSA feature extraction was tested on an invoice dataset that includes invoices with partial or missing invoice content data. These experiments closely represent the real production setup and it was found out that the proposed method works well in this setting as well.

Overall, the thesis presents the foundations of invoice processing with coding as part of it and goes into details of invoice content data. These basic ingredients will help in future steps of developing the coding prediction system as the data and systems are familiarized.

As expected, conducting the thesis provided numerous ideas for future research on the topic. Based on the small performance increases for account codes, the tested feature extraction methods are not able to fully retrieve the semantics similarly from the invoices. Therefore, more complex and recent feature extraction methods like language model embeddings could be able to capture the abstract semantics better. Also, more experimenting is needed to verify the assumptions and the optimality of the preprocessing methods proposed by this work.

If more complex feature extractors are taken into use there might be a benefit from training feature extractors with data from multiple companies as there probably are some similarities among the companies at least within the same industry. This would provide more training data to train the extractor and address the cold start problem so that coding predictions can be given without many historical datapoints. The similarities in the semantics are probably helpful in account code predictions, but for management dimensions, each company has different practices that need to be learned.

For a couple of datasets, the best results for management dimensions were obtained by feeding the most correlative words to target the classifier. This yields that for management dimensions some words strongly relate to certain coding values. One future development would be then to test if these related values could be found in the invoices and mapped to related coding values without using a classifier at all.

Utilizing the invoice content data unlocks the ability to predict multiple coding rows for one invoice. There exist multiple possible approaches to predict multiple coding rows for example feeding the invoice content features to the proposed machine learning pipeline one invoice line at a time. Therefore, the ideas and methods introduced in this work can be leveraged for predicting multiple coding rows.

Finally, it can be stated that this work was a good initial research of utilization of the newly enabled raw invoice content data for various tasks. It showed that there exist various statistical and machine learning methods to utilize the invoice content data source. In the future, there will be numerous other tasks leveraging the invoice content data to further automate the laborious task of invoice processing.

REFERENCES

- [1] Chiara Bardelli et al. “Automatic Electronic Invoice Classification Using Machine Learning Models”. eng. In: *Machine learning and knowledge extraction 2.4* (2020), pp. 617–629. ISSN: 2504-4990.
- [2] Yoshua Bengio et al. “A Neural Probabilistic Language Model”. In: *J. Mach. Learn. Res.* 3.null (Mar. 2003), pp. 1137–1155. ISSN: 1532-4435.
- [3] Hampus Bengtsson and Johan Jansson. “Using Classification Algorithms for Smart Suggestions in Accounting Systems”. MA thesis. Department of Computer Science and Engineering, Chalmers University of Technology, 2015. URL: <https://api.semanticscholar.org/CorpusID:55870797>.
- [4] Johan Bergdorf. “Machine learning and rule induction in invoice processing : Comparing machine learning methods in their ability to assign account codes in the bookkeeping process”. MA thesis. KTH, School of Electrical Engineering and Computer Science (EECS), 2018.
- [5] Piotr Bojanowski et al. *Enriching Word Vectors with Subword Information*. 2017. arXiv: 1607.04606 [cs.CL].
- [6] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. eng. In: *Proceedings of the 22nd ACM SIGKDD International Conference on knowledge discovery and data mining*. Vol. 13-17-. Ithaca: ACM, 2016, pp. 785–794. ISBN: 1450342329.
- [7] *Compliance with the European standard on eInvoicing*. URL: <https://ec.europa.eu/digital-building-blocks/sites/display/DIGITAL/Compliance+with+eInvoicing+standard>.
- [8] Scott Deerwester et al. “Indexing by latent semantic analysis”. eng. In: *Journal of the American Society for Information Science* 41.6 (1990), pp. 391–407. ISSN: 0002-8231.
- [9] Markus Esswein et al. “Improving Invoice Allocation in Accounting—An Account Recommender Case Study Applying Machine Learning”. eng. In: *Digital Business Transformation*. Lecture Notes in Information Systems and Organisation. Cham: Springer International Publishing, 2020, pp. 137–153. ISBN: 9783030473549.
- [10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [11] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*. 2010. arXiv: 0909.4061 [math.NA].

- [12] Niclas Hedberg. “Automated invoice processing with machine learning : Benefits, risks and technical feasibility”. MA thesis. KTH, School of Industrial Engineering and Management (ITM), 2020.
- [13] *International financial reporting standards*. Oct. 19, 2023. URL: <https://www.ifrs.org/> (visited on 10/19/2023).
- [14] Kamran Kowsari et al. “Text Classification Algorithms: A Survey”. eng. In: *Information (Basel)* 10.4 (2019), pp. 150–. ISSN: 2078-2489.
- [15] Quoc V. Le and Tomas Mikolov. *Distributed Representations of Sentences and Documents*. 2014. arXiv: 1405.4053 [cs.CL].
- [16] Christopher Lesner et al. “Large-Scale Personalized Categorization of Financial Transactions”. eng. In: *The AI magazine* 41.3 (2020), pp. 63–77. ISSN: 0738-4602.
- [17] Daniele Micci-Barreca. “A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems”. In: *SIGKDD Explor. Newsl.* 3.1 (July 2001), pp. 27–32. ISSN: 1931-0145. DOI: 10.1145/507533.507538. URL: <https://doi.org/10.1145/507533.507538>.
- [18] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. “Linguistic Regularities in Continuous Space Word Representations”. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Lucy Vanderwende, Hal Daumé III, and Katrin Kirchhoff. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 746–751. URL: <https://aclanthology.org/N13-1090>.
- [19] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. arXiv: 1310.4546 [cs.CL].
- [20] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].
- [21] Justin Munoz, Mahdi Jalili, and Laleh Tafakori. “Hierarchical classification for account code suggestion”. eng. In: *Knowledge-based systems* 251 (2022), pp. 109302–. ISSN: 0950-7051.
- [22] Tieto Finland Oy. *Release note: TEAPPSXML v.3.0*. Dec. 19, 2017. URL: https://bix.tieto.com/infoFiles/TEAPPSXML_v3.0_release_note_0.pdf.
- [23] Rasmus Berg Palm, Florian Laws, and Ole Winther. *Attend, Copy, Parse – End-to-end information extraction from documents*. 2021. arXiv: 1812.07248 [cs.CL].
- [24] Jeffrey Pennington, Richard Socher, and Christopher Manning. “GloVe: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: <https://aclanthology.org/D14-1162>.
- [25] Mary S. Schaeffer. *Accounts payable a guide to running an efficient department*. eng. 2nd ed. Hoboken, N.J: Wiley, 2004. ISBN: 1-280-34628-0.

- [26] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2016. arXiv: 1508.07909 [cs.CL].
- [27] Marina Sokolova and Guy Lapalme. “A systematic analysis of performance measures for classification tasks”. In: *Information Processing and Management* 45.4 (2009), pp. 427–437. ISSN: 0306-4573. DOI: <https://doi.org/10.1016/j.ipm.2009.03.002>. URL: <https://www.sciencedirect.com/science/article/pii/S0306457309000259>.
- [28] John. Stittle and Bob Wearing. *Financial accounting*. eng. SAGE course companions. Los Angeles, [Calif.] ; SAGE, 2008. ISBN: 1-4129-3502-4.
- [29] H Theil. *Economic forecasts and policy*. eng. 1961.
- [30] Gregory Valiant Tim Roughgarden. *CS168: The Modern Algorithmic Toolbox Lecture 9: The Singular Value Decomposition (SVD) and Low-Rank Matrix Approximations*. May 2, 2023. URL: <https://web.stanford.edu/class/cs168/l/19.pdf> (visited on 12/07/2023).
- [31] *VAT invoice requirements*. URL: <https://www.vero.fi/en/detailed-guidance/guidance/48090/vat-invoice-requirements2/>.
- [32] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. “On early stopping in gradient descent learning”. eng. In: *Constructive approximation* 26.2 (2007), pp. 289–315. ISSN: 0176-4276.

APPENDIX A: INVOICE XML

Example invoice in Figure 1.1 presented in Finvoice 3.0 format.

```

1 <?xml version="1.0" encoding="ISO-8859-15"?>
2 <?xml-stylesheet href="Finvoice.xsl" type="text/xsl"?>
3 <Finvoice Version="3.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:noNamespaceSchemaLocation="Finvoice3.0.xsd">
5   <MessageTransmissionDetails>
6     <MessageSenderDetails>
7       <FromIdentifier SchemeID="0037">00371234567890</FromIdentifier>
8       <FromIntermediator>NDEAFIHH</FromIntermediator>
9     </MessageSenderDetails>
10    <MessageReceiverDetails>
11      <ToIdentifier SchemeID="0037">00370987654321</ToIdentifier>
12      <ToIntermediator>003710948874</ToIntermediator>
13    </MessageReceiverDetails>
14    <MessageDetails>
15      <MessageIdentifier>123456</MessageIdentifier>
16      <MessageTimeStamp>2016-02-08T11:52:00</MessageTimeStamp>
17      <SpecificationIdentifier>EN16931</SpecificationIdentifier>
18    </MessageDetails>
19    </MessageTransmissionDetails>
20    <SellerPartyDetails>
21      <SellerPartyIdentifier>6666666-5</SellerPartyIdentifier>
22      <SellerOrganisationName>SEC oy</SellerOrganisationName>
23      <SellerOrganisationTaxCode>FI66666665</SellerOrganisationTaxCode>
24      <SellerPostalAddressDetails>
25        <SellerStreetName>Mannerheimintie 123</SellerStreetName>
26        <SellerTownName>Helsinki</SellerTownName>
27        <SellerPostCodeIdentifier>00100</SellerPostCodeIdentifier>
28        <CountryCode>FI</CountryCode>
29        <CountryName>Suomi</CountryName>
30      </SellerPostalAddressDetails>
31    </SellerPartyDetails>
32    <SellerInformationDetails>
33      <SellerAccountDetails>
34        <SellerAccountID IdentificationSchemeName="IBAN">FI2757800750155447</SellerAccountID>
35        <SellerBic IdentificationSchemeName="BIC">OKOYFIHH</SellerBic>
36      </SellerAccountDetails>
37      <SellerAccountDetails>
38        <SellerAccountID IdentificationSchemeName="IBAN">FI7429501800000014</SellerAccountID>
39        <SellerBic IdentificationSchemeName="BIC">NDEAFIHH</SellerBic>
40      </SellerAccountDetails>
41    </SellerInformationDetails>
42    <BuyerPartyDetails>
43      <BuyerPartyIdentifier>1234567-1</BuyerPartyIdentifier>
44      <BuyerOrganisationName>Basware Oyj</BuyerOrganisationName>
45      <BuyerOrganisationTaxCode>FI12345671</BuyerOrganisationTaxCode>
46      <BuyerPostalAddressDetails>
47        <BuyerStreetName>Hameenkatu 1</BuyerStreetName>

```

```

47     <BuyerTownName>Tampere</BuyerTownName>
48     <BuyerPostCodeIdentifier>33100</BuyerPostCodeIdentifier>
49     <CountryCode>FI</CountryCode>
50     <CountryName>Suomi</CountryName>
51 </BuyerPostalAddressDetails>
52 <BuyerContactPersonName>Jane Doe</BuyerContactPersonName>
53 <BuyerContactFreeText>Cost center: 6013</BuyerContactFreeText>
54 </BuyerPartyDetails>
55 <InvoiceDetails>
56 <InvoiceTypeCode>INV01</InvoiceTypeCode>
57 <InvoiceTypeCodeUN>380</InvoiceTypeCodeUN>
58 <InvoiceTypeText>Invoice</InvoiceTypeText>
59 <OriginCode>Original</OriginCode>
60 <InvoiceNumber>1002</InvoiceNumber>
61 <InvoiceDate Format="CCYYMMDD">20230201</InvoiceDate>
62 <OrderIdentifier>20130801</OrderIdentifier>
63 <AgreementIdentifier>12345</AgreementIdentifier>
64 <RowsTotalVatExcludedAmount AmountCurrencyIdentifier="EUR">1539.99</RowsTotalVatExcludedAmount>
65 <InvoiceTotalVatExcludedAmount
66     AmountCurrencyIdentifier="EUR">1539.99</InvoiceTotalVatExcludedAmount>
67 <InvoiceTotalVatAmount AmountCurrencyIdentifier="EUR">368.60</InvoiceTotalVatAmount>
68 <InvoiceTotalVatIncludedAmount
69     AmountCurrencyIdentifier="EUR">1909.59</InvoiceTotalVatIncludedAmount>
70 <VatSpecificationDetails>
71     <VatBaseAmount AmountCurrencyIdentifier="EUR">1539.99</VatBaseAmount>
72     <VatRatePercent>24,00</VatRatePercent>
73     <VatCode>S</VatCode>
74     <VatRateAmount AmountCurrencyIdentifier="EUR">24,00</VatRateAmount>
75 </VatSpecificationDetails>
76 <PaymentTermsDetails>
77     <PaymentTermsFreeText>Net 7</PaymentTermsFreeText>
78     <InvoiceDueDate Format="CCYYMMDD">20230208</InvoiceDueDate>
79     <PaymentOverDueFineDetails>
80     <PaymentOverDueFineFreeText>Overdue rate</PaymentOverDueFineFreeText>
81     <PaymentOverDueFinePercent>8,0</PaymentOverDueFinePercent>
82     </PaymentOverDueFineDetails>
83 </PaymentTermsDetails>
84 </InvoiceDetails>
85 <InvoiceRow>
86 <ArticleIdentifier>12345</ArticleIdentifier>
87 <ArticleName>Software expences towards product developmet</ArticleName>
88 <DeliveredQuantity QuantityUnitCode="kpl" QuantityUnitCodeUN="C62">12</DeliveredQuantity>
89 <InvoicedQuantity QuantityUnitCode="kpl" QuantityUnitCodeUN="C62">12</InvoicedQuantity>
90 <UnitPriceAmount AmountCurrencyIdentifier="EUR">10,00</UnitPriceAmount>
91 <UnitPriceNetAmount AmountCurrencyIdentifier="EUR">10,00</UnitPriceNetAmount>
92 <RowPositionIdentifier>1</RowPositionIdentifier>
93 <RowVatRatePercent>00,00</RowVatRatePercent>
94 <RowVatCode>S</RowVatCode>
95 <RowVatAmount AmountCurrencyIdentifier="EUR">00,00</RowVatAmount>
96 <RowVatExcludedAmount AmountCurrencyIdentifier="EUR">100,00</RowVatExcludedAmount>
97 </InvoiceRow>
98 <InvoiceRow>
99 <ArticleIdentifier>434355</ArticleIdentifier>
100 <ArticleName>License costs</ArticleName>
101 <DeliveredQuantity QuantityUnitCode="kpl" QuantityUnitCodeUN="C62">1</DeliveredQuantity>
102 <InvoicedQuantity QuantityUnitCode="kpl" QuantityUnitCodeUN="C62">1</InvoicedQuantity>
103 <UnitPriceAmount AmountCurrencyIdentifier="EUR">99,99</UnitPriceAmount>
104 <UnitPriceNetAmount AmountCurrencyIdentifier="EUR">99,99</UnitPriceNetAmount>
105 <RowPositionIdentifier>2</RowPositionIdentifier>
106 <RowVatRatePercent>00,00</RowVatRatePercent>

```

```
105 <RowVatCode>S</RowVatCode>
106 <RowVatAmount AmountCurrencyIdentifier="EUR">00,00</RowVatAmount>
107 <RowVatExcludedAmount AmountCurrencyIdentifier="EUR">100,00</RowVatExcludedAmount>
108 </InvoiceRow>
109 <InvoiceFreeText>Thank you for your business!</InvoiceFreeText>
110 </Finvoice>
```