Automated classification of receipts and invoices along with document extraction

Master's Thesis
University of Turku
Department of Future Technologies
Computer Science
2020
Roope Lehtonen

Supervisors:
Paavo Nevalainen
Mika Murtojärvi

UNIVERSITY OF TURKU
Department of Future Technologies

ROOPE LEHTONEN: Automated classification of receipts and invoices along with document extraction

Master's Thesis, 57 p., 4 app. p.
Computer Science
November 2020

Companies might receive dozens or even hundreds of receipts and invoices per day. It consumes a lot of working hours to keep them all organized – invoices must be paid on time and receipts must be archived properly. This research aims to reduce the amount of manual labor the organizing requires with automated classification.
Personally, I'm writing this thesis in collaboration with my workplace – a company called Eneroc Ltd. They had a problem with document classification consuming too many working hours. Therefore, they created a system to automate this process. The existing system uses a text-based approach that searches for specific key words in the documents. The system works rather well, but the company wanted to find out if some modern approach could outperform the existing system and add more features into the process.
The goal of this research is to find out if a machine learning based approach could be used to classify documents into invoices and receipts. In addition to the classification, the approach should also be able to collect key information from the documents. This thesis describes the workflow of creating a machine learning based solution to tackle the given challenge.
The research resulted in an application that takes in invoices and receipts in PDF format. The system trains a k-nearest neighbors model with training data, that was created in the process of the research. The model is then used to classify different parts of the new PDF files into predefined categories. The key information is extracted from these categories.
The k-NN model was validated with k-fold cross-validation. The validation showed that the model is performing correctly. Some preprocessing was also introduced in the process, which further improved the results. Good results with the k-NN model imply that using a proper machine learning solution would be profitable.
The final classification between receipts and invoices, as well as the key information extraction, is done based on the classified document parts. This works rather well on the classification and simple key information extraction. But more complex key information extraction – like the product list extraction – still requires more work.
The research proved that machine learning solution could be used to classify documents into invoices and receipts, and also to collect key information from the documents. The created application isn't yet ready for deployment, but it gives a good foundation for future development. The research also shows which steps to take next and where to focus on when improving the system.

Keywords: Classification, Machine learning, k-nearest neighbors, Portable document format, Document extraction

Contents

List of Tables

1 Introduction

Companies might receive dozens or even hundreds of invoices and receipts per day. And almost all of them look different, whether in electric or in paper format. It takes quite a lot of working hours to keep them all organized, and to find the relevant information from them. This thesis aims to tackle the problem with computer automation.

This thesis describes the work flow of creating a new system for processing invoices and receipts. The system should be able to identify these documents as invoices or receipts. Furthermore, it should also be able to identify relevant key information from the documents. This information includes, for example, the total amount and the tax percentages.

The next section describes the inspiration behind this thesis – why is the system created, and how it will be used. The section after that will guide you through this thesis. It describes the main structure of the thesis, and presents which topics will be discussed in the later chapters. It will also tell you what knowledge you should have before reading the thesis.

1.1 Inspiration

I work at a company called Eneroc Ltd. They had a problem with receipts and invoices consuming too many working hours. So instead of employees reading through the documents, they developed a piece of software that does the work.

This current approach handles the whole document as one long line of text. It searches

for a set of predefined keywords in the text. When it encounters such a word, it tries to find some relevant information from the next or previous words. For example, when the system finds a keyword *"VAT"*, it tries to find some number nearby that could be the VAT percentage. This system works rather well on finding simple information, like the VAT percentage or the total amount. But it doesn't know how to read complex information, like the receipt items.

The company wanted a new, improved method to handle the incoming invoices and receipts. Today's technologies allow the use of machine learning and other advanced methods that could help to improve the old approach.

1.1.1 Existing solutions

There are already multiple solutions available on the market that offer similar functionalities as the new system should have. But it's difficult to find a system that offers all the needed features. And it's impossible to find a system that can be as widely modified as your own system.

The existing solutions usually offer document scanning with some key information extraction. This data can then be stored into a cloud service or exported from the application. Many of the applications are designed only for receipt tracking, but there are also systems that support receipts and invoices. This kind of system could maybe be used to achieve the desired results. But such a system would most likely require subscription payments. The existing applications also most likely contain multiple unnecessary features. And even with different payment plans, users most likely end up paying for features they don't use.

Creating an own system gives us the ability to include only features that are needed – and new features can also be added later. In the own system, the only cost is the development process. After the development, it will be very cheap to maintain the system. Creating an own system also gives the opportunity to sell product licenses to other com-

panies and users once the system is working – potentially making the system profitable.

1.2 A guide for the reader

This thesis handles topics that include classification, machine learning, k-fold cross-validation and k-nearest neighbors algorithm. The thesis will guide you through the topics briefly, but any prior knowledge in the topics will help you understand the actual content easier. This thesis is written for anyone interested in the work flow of creating a machine learning application. Or anyone interested in extracting information from documents.

By now you should have read – or at least glanced through – the table of contents. The backbone of this thesis consists of the chapters 3 (*Approach*) and 4 (*System description*). The former of these chapters describes the approach used to create the system, without going too deep into the technical details of the implementation. This chapter also provides an introduction into the used techniques. The latter chapter, on the other hand, describes the technical implementation of the system. It shows the used programming language and the essential third party libraries. This chapter assumes you are already familiar with the approach and the used techniques.

You should also take a look at the appendix A, before proceeding to the next chapter. This appendix contains an example invoice that will be referred to in multiple places throughout the text. The example invoice is an actual invoice from the data sets that are used in this thesis. The invoice is written in Finnish – like most of the other documents this thesis utilizes – but you will still see the basic structure of the invoice, even if you don't understand Finnish language. The most important information from the invoice is listed also in English in the appendix.

2 Targeted data

The data used in this research consists of receipts and invoices. Most of the documents are PDF files, which will be the main focus for this research. In addition to the PDF files, also XML and traditional paper documents are included in the data. The final system should be able to handle any of these file types, and transform them into one solid form which can then be further processed. All of the documents are machine written. Hand written documents are excluded from this thesis.

This chapter briefly tells about the different data types and how they should be handled. It also describes what kind of data is used in the research and how much data is available. Later in section 4.1 you will learn how each of these data types is processed.

2.1 Supported data types

2.1.1 XML files

XML stands for e**X**tensible **M**arkup **L**anguage [1]. A markup language doesn't contain any styles, but instead shows the structure of a data. For the purpose of this thesis, XML files are very easy to work on. We can just pick the correct values from the data structure as long as we know the attribute names we are looking for.

2.1.2 Paper format

Paper format is luckily stepping aside and making way for new electric formats, but especially receipts are still very often received in paper format. These documents must be turned into digital format in order to process them.

The first step is to scan the documents into images or PDF files. But this alone isn't enough. We must also read the text content on the documents using a method called **OCR**. OCR stands for **O**ptical **C**haracter **R**ecognition. Using OCR we can turn scanned documents into digital PDF files, where each recognized character is turned into a digital letter. This text can then be processed, for example copied or modified. Furthermore, OCR also preserves the location of the texts in the document. This proves to be very useful in the latter chapters – compared to reading the whole document in a one long string of text.

2.1.3 PDF files

PDF (**P**ortable **D**ocument **F**ormat) is a file format developed by Adobe Systems Incorporated, and it is currently ISO standardized. It aims to be a format that can be created and viewed in multiple different environments. These files may contain for example text, graphics and hyperlinks. [2]

PDF files are the main focus of this thesis since majority of receipts and invoices are received in this format. The diversity of PDF files also helps in achieving the desired results. PDF files can contain images and texts in different sizes and colors. There isn't any template for PDF invoices or receipts – each company has their own template for these documents. This is the key reason for this whole thesis. The final system should be able to process receipts and invoices of any template.

2.2 The data used in the research

The data used in this research consists of receipts and invoices. These documents are actual documents that Eneroc Ltd. and it's associates have received. The total amount of the used documents is 336, which consists of three different data sets. The first one has 60 files, the second one 52 files and the third and the largest data set has 224 files. The first two data sets are provided by Eneroc Ltd. and the third one by it's associates. The example invoice in appendix A is taken from the second data set.

Fortunately, the range of the documents is rather large. The receipts contain for example taxi receipts, parking receipts, grocery store receipts and software subscription receipts. The invoices contain for example rent invoices, magazine invoices and electric equipment invoices.

On the downside, the variety in the documents is pretty limited. Many of the documents come from the same senders and may even contain the same items. For example, monthly billed internet access invoices are more or less copies of each other.

3 Approach

When the research began, the best approach wasn't clear right away. Thus, another approach was first tested. This approach is called **template matching**. Template matching is used to find predefined patterns in images. This method was however quickly abandoned because of the diversity of the receipt and invoice documents. Such templates don't exist that would work on every invoice and receipt. It was clear that handling text as text is the best approach.

In order to find all desired information from the documents, a classification approach was introduced. In this approach, the documents are cropped into multiple smaller areas and these areas are then classified into predefined categories. For example, all total amount related information would be classified into "*Total amount*" category. In the end, all useful information is classified into correct categories. The desired information – for example the total amount – is then extracted from these categories.

In this chapter, I will first briefly tell about the template matching technique and the reasons why it was abandoned. Then I will move on to the techniques, the actual approach and the programming methods that were used. This chapter focuses on describing the approach without going into the technical details of the product. The next chapter 4, "*System description*" , will tell the technical details and show how the approach, described in this chapter, is implemented.

3.1 The discarded template matching method

This chapter explains the basic idea of the template matching method, how it was implemented for this task, and why it didn't work well for this research. "Template matching" as a term already gives a hint of what template matching is. In template matching, we have a predefined template. Then we try to find occurrences of the template in new, unforeseen data. Template matching could be used, for example, to find faces in images [3]. In that approach, the template would be an image or images of a face. Then the system would try to find the best position in the image where the template matches with the image – resulting in a working face detection system.

In this research, the idea of using template matching, was to find elements in the documents that are typical for invoices and receipts. For example, in invoices the total amount is almost always stated somewhere in the document. Therefore, we could have a set of templates taken from actual occurrences of total amounts in invoices. Then we would try to find matches for these templates in the new documents.

This task, of finding the total amount with template matching, was actually implemented in the early stages of this research to see if the method could work. Thus, it will be used as the main example in this section. The implementation used the total amount template from 10 different invoices with good variety. Then it tried to find similar templates in new invoices to find the total amount.

The problem with the template matching approach in this research is the wide variety of the document templates. If we take a template of the total amount from one invoice, it would probably work rather good on new invoices that appear in the exact same document template. But it would most likely fail on new invoice templates, since the template for the total amount could be very different. This method would work good only if there were some standardized invoice template that everyone follows.

Furthermore, the total amount changes on different invoices. So, including the total amount in the template means that one template from an invoice wouldn't match com-

pletely with an invoice of the same format, if the total amount differs. If we don't include the total amounts in the templates, then how do we know where the total amount is located on the document?

3.2 Processing text as text

Since invoices and receipts are – most of all – text documents, it felt more natural to process them as text rather than using image based template matching. At this point it was very clear that this is the next direction of the research. Even if some of the documents are scanned paper documents in image format, it would be extremely useful to first turn them into text format and then process them as text documents. This is where optical character recognition (OCR) steps in.

Optical character recognition is a method of turning text in image format into a machine readable text format [4]. These methods usually utilize machine learning techniques, and they can be very complex systems. This research utilizes two different systems: **Tesseract** and **ABBYY Finereader**. These systems and OCR are better described in the section 4.1.2.

The earlier example, of finding the total amount in a receipt, becomes much more simpler with a text based document. Now, we don't have to find any predefined image template in the document. Instead, we try to find predefined words in the document. The format of the document is no longer relevant. In order to find the total amount, we would search for phrases like "*Total*", "*Total amount*", "*Sum*", etc. and try to find an appropriate sum nearby.

3.3 Tools

3.3.1 Programming language

For me, personally, there was only one answer to what programming language should be used. The answer is **Python**. First of all, I already had some experience with Python. Mostly in text-based machine learning tasks – tasks, that involve similar techniques, as this thesis would most likely use. Secondly, Python offers powerful tools without complex programming structures. This is the ideal situation for building small reusable scripts – such as this thesis would probably use.

Python has also a great collection of useful libraries. This research uses multiple libraries to, for example, identify relevant information and tag them. And ultimately, Python has also a comprehensive selection of easy-to-use machine learning libraries.

3.3.2 Machine learning and k-NN

My workplace already has an existing solution that reads receipts and invoices to find all relevant information. This existing system works with predefined keywords to find that information. They wanted to see if the results could be improved with more modern techniques. Therefore, machine learning was in key position when the approach was planned.

Machine learning methods can be quite complex and heavy. Sometimes it's easier to start with a simpler technique, and if it shows significant results, move to a real machine learning solution. In this research, the simpler method would be **k-NN**, which is better described in the following section. Eventually, the whole machine learning part was dropped out of the scope of this research. The scope is to see if machine learning methods could improve the results of the earlier system, and k-NN already gives us a good clue of the machine learning capabilities.

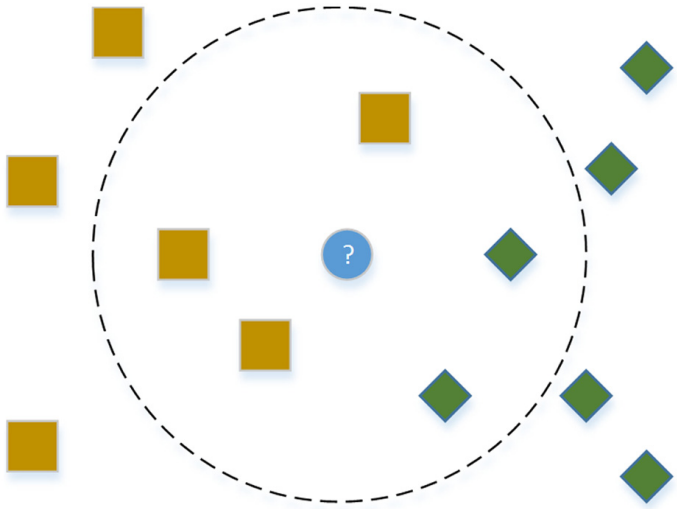

Figure 3.1: Illustration of k-nearest neighbors. [5]

k-NN

k-NN stands for **k**-**N**earest **N**eighbors. This is an approach where new data is classified by looking at earlier data, that has already been classified. We try to find predefined amount (k) of data samples in the classified data, that are the most equal samples with the new data (Nearest Neighbors). Then we classify the new data into the most popular class among the nearest neighbors.

To better understand the k-NN functionality, take a look at the figure 3.1. In this example image, the yellow squares on left are already classified into one category and the green diamonds on the right to another category. The blue circle in the middle is a new sample that is classified based on the 5 nearest neighbors. The dashed line shows that 3 squares and 2 diamonds are the 5 nearest neighbors. Based on this amount, the new sample would be classified as an orange square.

In reality, k-NN isn't a machine learning method, since the model doesn't actually learn anything. Instead it works by making decisions solely based on the old data. Nevertheless, if k-NN provides us with significant results, it is very likely that machine learning methods would provide significant results as well.

3.4 The classification task

One of the main tasks of this research is to be able to classify new documents into categories. The most important classification task is to correctly identify a document as an invoice or a receipt. It is important for companies to pay invoices on time and to correctly archive receipts, so correct classification is crucial.

Other classification tasks of this research include categorizing different parts of the documents into predefined categories. For example, all tax related information should be categorized into "*Tax information*" category. This will help us in identifying the most relevant information on the document. For example, once we have all the tax related information in one group, it will be much easier to read the tax percentages from that group.

Classifying the document parts into categories means that it's not enough to just inspect the documents as a whole, but instead they should be inspected bit by bit. The idea for this is to go through the text in narrow rows and columns. Then these small areas are classified into six predefined categories:

1. Receiver / Sender information
2. Product information
3. Total amount information
4. Tax information
5. Receipt information
6. Invoice information

All text, that doesn't belong to any of the above categories, is classified as useless information. So, after the classification process is done, we should have only useful information in the correct categories.

The first category, "*Receiver/Sender*", will contain all information about the sender and the receiver. This information will then be used to identify the sender by filtering out the receiver information. The second category, "*Product*", contains all individual items from the document. This class also contains the pricing information for the individual products. The third category, "*Total*", contains information related to the total amount. From this class we will pick the total amount of the receipt or the invoice. The fourth class, "*Tax*", contains information about the taxing. This includes the total tax percentage, and also the tax percentages for individual items, if such exists. The fifth class, "*Receipt information*", contains all information related to receipts – for example, the used payment method. The sixth class, "*Invoice information*", on the other hand, contains information related to invoices. This includes for example the bank account number, where the invoice is paid. The last two categories, Receipt information and Invoice information, are also used to determine whether the document is a receipt or an invoice.

3.5 Preprocessing

Wherever machine learning is involved, there is preprocessing. Preprocessing includes all the steps where the data is processed so that machine learning can learn something from it. Preprocessing is also done to improve the learning and the results [6].

In this research, it became very early clear that some preprocessing is needed. The information in the documents is very different on each document, so it cannot be given to a machine learning model as is. The following subsections describe what preprocessing methods were used in the system, and which methods could be used to further improve the results.

3.5.1 Text cleaning

One of the first and easiest preprocessing methods that can be done is turning all letters into lowercase. This makes sure that the same word in two different cases isn't treated as two different words. For example, many receipts might say "TOTAL" in uppercase letters to make it stand out better. We don't want to treat this any differently than "total" in lowercase.

More sophisticated preprocessing methods were also introduced in the application. These methods are **lemmatisation** and **stop word removal**. These methods are language specific, which means that we have to figure out the language for each document before processing them. These methods and their advantages are briefly explained in the next sections.

Stop word removal

Stop word removal is a method of removing common small words from text. In machine learning, this helps the model to focus on the most important content. For example, with a sentence "*The total amount is $4.99.*", stop word removal would remove the words "*The*" and "*is*". The resulting sentence "*total amount $4.99*" still contains all necessary information without the irrelevant content.

Lemmatisation

Lemmatisation is a process where words are transformed into their base format. This helps machine learning models to process the different formats of one word as equal. Lemmatisation would turn for example the words "*tax*", "*taxes*" and "*taxing*" into the base format "*tax*".

Sophisticated preprocessing problems

The sophisticated preprocessing methods also introduced a few problems. Firstly, some of the documents contain the same information in English and Finnish. This means that lemmatisation and stop word removal should be used for both languages. Especially lemmatisation for two languages doesn't really work. It is also difficult to find a working lemmatisation solution for Finnish language. Therefore, Finnish language doesn't use lemmatisation in the produced application. In the final application this should be fixed to further improve the results.

3.5.2 Tagging

We don't want to teach any machine learning model the specific information in the documents. Instead, we want to teach the general format and patterns of the documents. This way, the model will be more successful when processing new documents.

Let's examine the example invoice in appendix A to get a better idea. If we teach a machine learning model that for example "*Yhteensä € 34,89*" describes the total amount. Then we give it a document that contains a text "*Yhteensä 41,20 €*". The model might not identify this as a total amount field, since they contain different numbers. To overcome this problem, we use a method called "tagging".

Tagging is a method where relevant, changing information is replaced with predefined strings. These strings should be something that don't appear in the text naturally. In the above example, we could replace the changing numerical amounts to a tag "<*amount*>". So, instead of the original text, we would teach the machine learning model that "*Yhteensä € <amount>*" describes total amount. Then the new data – preprocessed to "*Yhteensä <amount> €*" – would much more likely be identified as a total amount field. Other fields that could be tagged are for example emails, organizations and phone numbers.

3.6 Creating training data and putting it into use

One big issue with machine learning tasks in general, is the lack of data. All machine learning solutions need training data in order to learn anything – and the more the merrier. Unfortunately, good training data is sometimes hard to come by. And even if data was available, it might not be suitable for the machine learning task. Very often researchers have to rely on gathering the suitable training data themselves. This applies to this research as well.

Eneroc Ltd. and it's associates provided me with about 300 documents. These documents are actual receipts and invoices, that the companies have received in their work. And the final application would be used with very similar documents. It is good to have real data to be used as the training data, but unfortunately this data isn't classified at all.

3.6.1 Classification application

In order to create a machine learning model that knows how to classify documents, and parts of documents, into predefined categories, we need training data that is already classified into these categories. Therefore, an application was created, that is used to classify the data into these categories manually.

The application shows one document at a time to the user, and goes through the document in small areas. It shows where the area is on the document, and it also shows what text was extracted from that area. User then has the option to classify each of the areas into one of the predefined categories, presented in the section 3.4. As a result, we will have two text files per each document. One of these files contains all extracted texts, each on a new line. And the other file contains the labels i.e. the classes, that the lines belong to. A more specific description of the application can be found in the chapter 4.3.

The classification process takes quite some time to complete, since all the available documents should be classified, in order to have as large training data set as possible. And

classification by hand is a slow process, even when done with the classification application.

3.6.2 Training the model

Once we have the training data in a suitable format, we can move on to training the machine learning model. At this point, the data is in the exact format that was extracted from the PDF files. So, before training the model, the data is preprocessed as described in the section 3.5. This allows the model to learn the common patterns of the documents, instead of some specified details.

Before training the model, the data must be divided into training and test data, in order to validate the model. The model should never be tested against data that it has already seen. This would give us unreliably good results, since the model has learned how the test data should be classified. Therefore a separate test data is used to test the model accuracy. One unfortunate fact is, that some of the documents in the available data are very similar in format and content. So, the training data and the test data will most likely overlap each other a bit, for some of the documents.

To overcome this problem, and to get reasonably reliable results, a method called **k-fold cross-validation** is used. In all its simplicity, this means that the model is split into training and test data sets multiple times, and the model is trained and validated with all these sets [7]. The final accuracy is then measured from the average of all accuracies. This way, if one of the loops – called *fold* – provides too good results, the other folds will compensate for it.

The figure 3.2 visualizes how the k-fold cross-validation works. This example image uses 10-fold cross-validation, which means that the data set is split into training and test data set 10 times. Each fold uses 1/10 of the data set as the test data set (or validation data set) and 9/10 as the training data set. The test data sets never overlap each other between the folds. This way, each data sample is used as a validation sample exactly once.

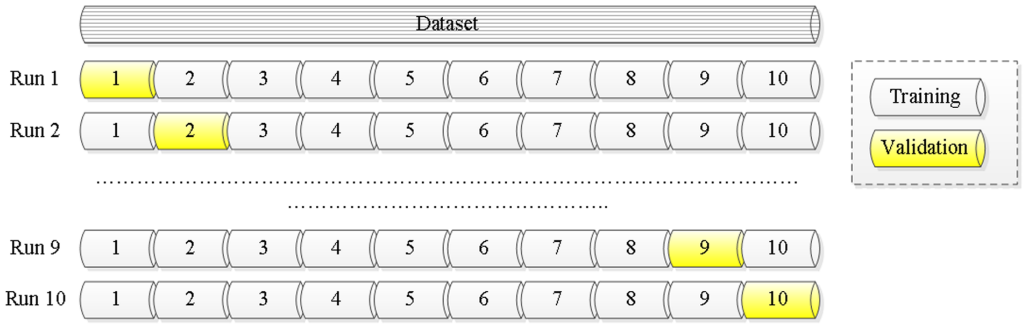

Figure 3.2: Flowchart of 10-fold cross-validation. [5]

The test data and the folds are only needed for validating the system. In the final product, all available training data can be used in the training. This is because all the new documents, processed with the final system, will be unforeseen data.

3.7 Reading the classified information

After the model is trained, and documents have been processed with it, we will have data that is split into the predefined categories. Now, we still must extract only the needed information from the classified texts.

The first class contains information about the **sender** and the **receiver**. This includes names, cities, countries, postal codes, companies, etc. In order to distinguish the sender information from the receiver information, the system needs to be provided with the receiver information. Then it tries to remove all receiver related information from the extracted text, and leave only unique text.

Reading **product information** is by far the most difficult category, since it can contain pretty much any information. And on the other hand, it is also the least essential of the categories. Therefore, little effort was put into reading the product information. Currently it just removes duplicated information from the extracted text. And user must then do the final decision on which of the texts are actual products. This category is something that should be improved in the future. Chapter 6 will talk more about future improvements.

The **total amount** is a very essential piece of information on each receipt and invoice. The system will therefore first try to find it in the extracted text. And if it isn't found, the system tries to calculate it from the tax percentage and the other found amounts.

Similar to the total amount, the **tax percentage** is also rather essential information. And it is also possible that there are more than one tax percentages on one document. For example, food products might be taxed differently than alcohol products. And it is important to find all used tax percentages and preferably the amounts that are taxed with each percentage. If tax percentages or taxed amounts aren't found, the system will also try to calculate these from the total amount and the other amounts and percentages.

Last but not least, the system will try to determine whether the processed document is **an invoice** or **a receipt**. For this purpose, the two last categories are used. These categories contain information common for an invoice or a receipt. In addition to determining the document type, the system will also try to find other essential information from these categories. This additional information together with the amount of data in the categories is then used to determine the document type. For example, a document with a due date, is most likely an invoice that requires action from the user.

In the end, the system will print all the gathered information, described in this section, to the user. It will also provide the user with possible errors that occurred during the process. For example, if the tax percentage information is missing completely, it is impossible to return any tax percentage – found or calculated. The system will also tell the user which information is found on the document and which information is calculated.

4 System description

The previous chapter, chapter 3, described the approach that was used in the research of this thesis. This chapter will now dive deeper into the technical details and the code that was used to achieve the described approach. The structure of this chapter will somewhat follow the structure of the previous chapter.

The first section in this chapter describes how the different data types, presented in the section 2.1, are processed. It concludes why the PDF file type became the most important data type in the research. The latter sections describe how the PDF files are processed – how they are turned into training samples, how the training samples are used to classify new documents and how to extract the wanted information from the new documents.

4.1 Processing different data types

This chapter briefly tells how each supported data type is processed. The possible types for the documents are XML files, paper documents and PDF files. You will see why the XML files were eventually bypassed in the research, how paper documents are processed and why the PDF file type is paid so much attention to.

4.1.1 Processing XML files

XML files are markup files, so extracting information from them is rather easy. You just need to know the correct tag names for the information you are searching for. And then,

you of course need a tool to extract the information. Python has a package called `xml`, that contains a set of tools that can be used to process XML files. This research utilized a submodule `xml.etree.ElementTree` to extract the wanted information.

It was concluded that extracting the information from XML files is very easy. Therefore, the XML files were eventually dropped out of the scope of this research. The research aims to find some predefined information from a given document – e.g. tax percentage from an invoice. And if it's as easy as reading a tag `<VatRatePercent>`, it is not really worthwhile to invest time in this topic.

4.1.2 Processing paper documents

Almost all documents that a company receives in paper format, are scanned into digital format, for easy storage and to ensure that the documents don't disappear. Scanners usually provide the option to scan a document into a PDF file or an image format. Unfortunately the PDF files usually don't include any actual text content. And image formats obviously don't contain any text. Before further processing, these file formats must be transformed into files that contain text.

The image format documents are machine read into PDF files with the help of Optical Character Recognition. Some scanners provide built in functionalities to turn scanned documents into PDF files with OCR [8]. If this is the case, the files should be scanned into PDF directly. But since the option is not always available, this research utilized two systems for the OCR task: **Tesseract** and **ABBYY Finereader**.

Tesseract is an open-source OCR engine that can be downloaded and used directly from the command line. It was tested on a receipt from a grocery store. Tesseract performed pretty good, but there are some errors in the text output. These errors in the text would have a negative impact on the machine learning model training and validation.

ABBYY Finereader on the other hand, is a paid OCR engine. My workplace, Eneroc Ltd., is already using this system to read scanned documents into PDF format. Therefore,

ABBYY Finereader was also tested for this research. The same receipt scanned with Tesseract, was now scanned with ABBYY Finereader. The results seemed much better, so ABBYY Finereader would probably be used in the final product.

After the paper documents are turned into PDF files, they will be processed just like any other PDF document. The actual method used in the conversion isn't relevant. Therefore this topic wasn't studied any further.

4.1.3 Processing PDF files

PDF is one of the most common invoice and receipt file formats received by companies – if not the most common. Therefore, most of the research focuses on extracting information from PDF files. There are countless amounts of different templates that PDF files come in, so the information extracting isn't as trivial as with XML files. PDF file format is also rather complex, so the text cannot be extracted directly from the document file.

The PDF files will be completely processed with Python. Python has packages that can be utilized, for example, to extract the text from the files. The following sections will show in detail how the PDF files are processed in the research.

4.2 Turning PDF files into training data

In order to use the PDF files in a machine learning system, the files must be first turned into a suitable format. The object is to divide the files into smaller pieces and then classify each of these pieces into a predefined category.

The first approach was to turn the PDF files into images, and then split these images into smaller pieces. The pieces would then be read with OCR to extract the text from each piece. This didn't work as intended, so another approach was used. The next approach splits the actual PDF file into smaller PDF files to make sure the text remains as is. The following sections describe both of these approaches.

4.2.1 The discarded cropping of images

In the first approach the PDF file was first turned into an image. Then the image was cropped into smaller areas in a loop. Each of these crop areas was then read into text with the help of Tesseract. These texts could then be classified and used to train the machine learning model. This approach, however, had multiple problems, that would be very hard to solve. Therefore, the approach was discarded from the final approach.

Problems with the image cropping

The first problem was, when the document was cropped into narrow horizontal slices, the intersection would very often cut lines of text into two halves. This made it almost impossible for any OCR engine to actually recognize what the lines said. It is very difficult even for a human being to read these lines. And to further complicate the problem, the OCR machines would still recognize the split lines as text. The lines would be read into totally meaningless strings of characters. The figure 4.1 shows an example where split lines and complete lines have been processed with Tesseract. The figure helps to understand the problem with split lines. To overcome the problem, we would have to somehow recognize where the line spacing is located before cropping the image. With variations in font size, this would be very difficult as well.

Secondly, receipts and invoices are very often divided into some sort of columns. So, reading a full width row of text might belong to two different classes – for example tax percentages on the left column and total price on the right column. This means that the cropped areas should also be read in columns. With the current approach, this would sometimes lead into cutting lines of texts vertically half, which makes the classification very difficult.

Combine the horizontal cutting of texts with vertical cutting, and no one will know what the original document said. Once again it was concluded, that text files should be processed as text files.

Figure 4.1: Example from an invoice that has been split to narrow image rows. These rows have then been processed with Tesseract OCR engine. The grey area shows how Tesseract has processed the whole lines correctly, but the split lines produce completely meaningless text.

4.2.2 Cropping the PDF files

In order to crop the document, and to get reliable text output from the cropped area, it was decided to crop the actual PDF file and then extract text from the cropped file. This is ideal approach for the original PDF files, since the original text is kept intact, and we don't have to rely on OCR engines at any point. And if the original documents are in image format, we can use the OCR engines on the whole document rather than on a small area.

Cropping a PDF file isn't as easy as image cropping, since the PDF file type is rather complex. A couple different PDF Python libraries had to be tested and utilized to finally get the cropping to work. Some of the libraries cropped the PDF files only into images. And some of them cropped the PDF files correctly, but the files still included all text content from the original file.

The final approach was able to crop the document area to desired size, and then copy the text inside that area. Now this had to be done multiple times for the same file in different coordinates. The system reads the PDF document area size in pixels and divides it to predefined sizes. Then it goes through the file in an area of that size. For example 3 columns and 100 rows would use 1/3 of the original width and 1/100 of the original height

as the crop area size. The texts from each area can be written to a text file for example.

4.3 Classification application to produce training data

Once it was figured out how a PDF file can be cropped into smaller areas, and how the text can be extracted from each area, it was time to create training data for the upcoming machine learning task. It was very clear that the training data must be created by hand since no appropriate data exists yet. To help speed up this process, an application was created that can be used to classify the texts.

Python is not an ideal programming language to create graphical user interfaces. But since all other functionalities are programmed in Python, the application was also created with Python. Also, this will not be a commercial program, so the UI appearance is irrelevant. Python has a popular GUI package called `TkInter`, that was used to create the GUI.

The application takes in a folder of PDF files as an argument. Then it loops through the files, and all pages in each file. The GUI shows an image of the current page to the user. Then the application starts cropping the page and going through each crop area as described in section 4.2.2. The current crop area is then painted as a rectangle on top of the page image. The extracted text from this crop area is also shown to the user as plain text. User then has the option to classify the shown text into one of the six predefined categories, listed in section 3.4, by clicking the corresponding button. There are also buttons to skip the current crop area or to skip the whole page.

The figure 4.2 shows the layout of the application. The example invoice from the appendix A is currently being processed in the figure. On the left side, is an image of the processed document. And on top of the image – in the upper left corner – is a narrow rectangle showing the location of the current crop area. The extracted text from this crop area is visible in the upper right section of the application. All application buttons are

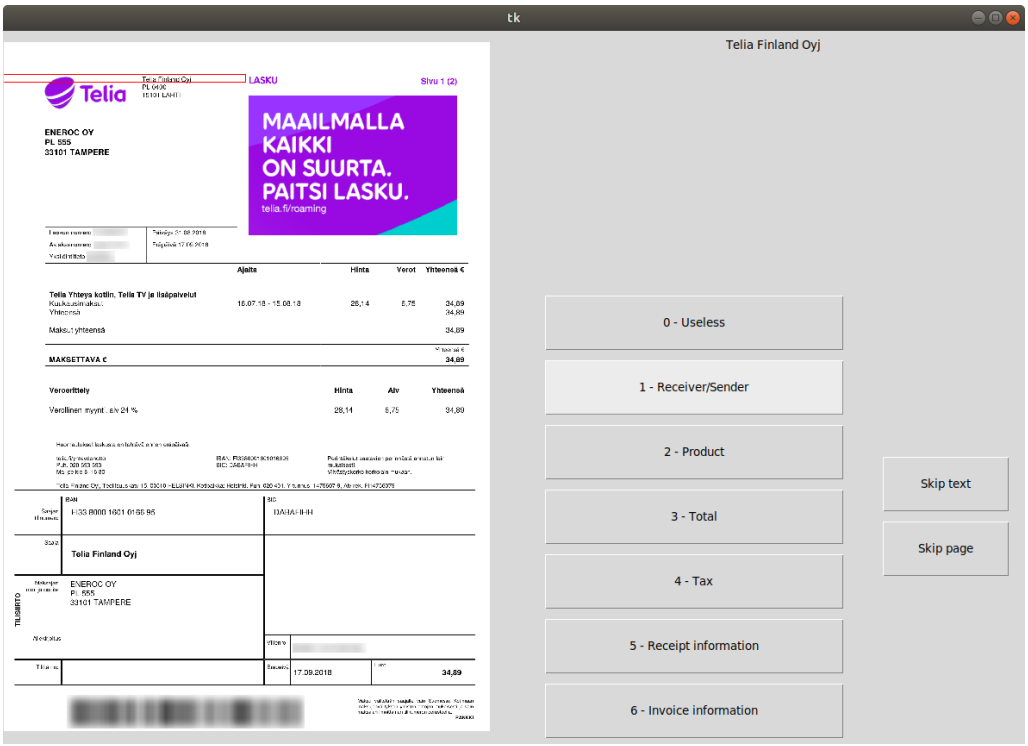

Figure 4.2: Graphical user interface of the manual classification application.

positioned below the text area. The buttons can be used to classify the current text into a category or to skip the current area or the whole page. In this figure, user should click the button "*1 - Receiver/Sender*", since the text "*Telia Finland Oyj*" is the name of the sender. The button is being hovered in this image, and is thus highlighted.

As a result, the application will produce two folders: "*texts*" and "*labels*". Both of these folders will contain one text file for each processed PDF file. The first folder, "*texts*", holds files that contain the extracted texts from each file – one area per line. And the folder "*labels*", on the other hand, contains matching labels for each line of each file in the texts folder. For example, a PDF file called "`./file.pdf`" would produce files "`./texts/file.txt`" and "`./labels/file.txt`". Then the first line of text in "`./texts/file.txt`" is the extracted text from the first processed crop area of "`./file.pdf`". And the first line in the file "`./labels/file.txt`" would contain the label (0-6) for the extracted text.

To better understand the application process, take a loog at the figure 4.3. This diagram shows how the application will go through the provided files, the pages in these files and the crop areas in each page. Whenever an area with text content is encountered, the program will wait for user input before proceeding. Based on the input, the area is either classified or skipped, or the whole page is skipped.

4.3.1 Problems of the classification application

As mentioned earlier, the structure of a PDF file is very complex. Therefore, handling PDF files is very error-prone. If one PDF file is processed without errors, it doesn't mean that all PDF files will work with the same process. This application needed a lot of refactoring to make sure it has proper error handling to process as many files as possible. The more training data we have, the better results we will have.

There were multiple different problems encountered in the files during the processing, some of which are listed here: When extracting text from the file, some files included an addition euro sign (€) at the end of each area, even though not visible on the PDF file. Some files contained characters that weren't recognized by the text extracting library. Some files showed all text content in each cropped area. And some of the files used an encryption method that isn't supported. These problems might be problems in the PDF files themselves, but more likely they are problems of the used PDF libraries.

Fortunately, most of the files processed correctly and resulted in the desired output. However, when the cropped areas were classified by hand, it wasn't always clear which class an area belongs to. Sometimes the areas contained parts of two classes, and sometimes the area contained data that is difficult to classify. For example, a text "*0.00*" contains very little information to any category. For these reasons, the button to skip a crop area was added into the application.

The later chapter 5.1 will show in detail how many of the available files were read and processed successfully. It shows that most of the skipped files are files that don't

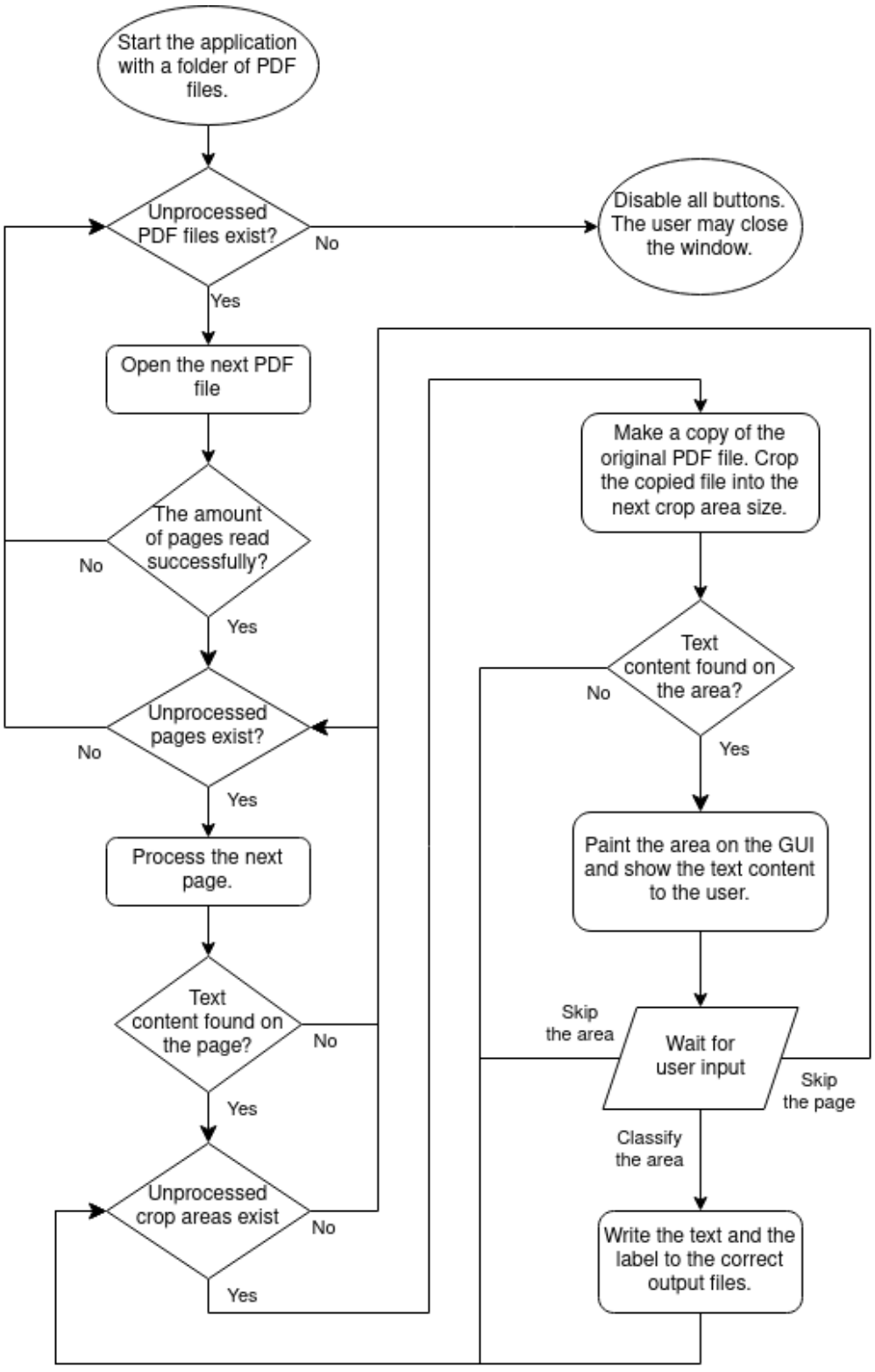

Figure 4.3: A flowchart for the training data creation application.

contain any text content. These are files that should be processed with OCR in the final application. The section 5.1.1 will also show how many training samples the available PDF data produced.

4.4 Training the k-NN model in action

Once the training data was created, it was time to put it into use. This means training the k-NN model with the training files, and then processing new files with that model. In order to see if the system really works, a k-fold cross-validation system was implemented. The validation showed promising results for the data as is, but the results could be further improved with preprocessing.

4.4.1 Training a k-NN model with Python

Training a k-nearest neighbors model in Python is relatively easy – thanks to comprehensive Python libraries, that is. This thesis utilized a package called `sklearn` (scikit-learn) for the training task. The package provides easy-to-use fitting functions with automatic stop word removal. User doesn't have to know anything about what happens behind the scenes.

More precisely, this thesis used a package called `TfidfVectorizer` from `sklearn` to vectorize the training texts. In short, this applies more weighting for the more important words, and less weighting for the less important words in the training data [9]. Then the model was trained with the vectorized texts and the label files together. The training utilized a package called `KNeighborsClassifier` from the `sklearn` package.

4.4.2 k-fold cross-validation with Python

The k-fold cross-validation method is shortly described at the end of the chapter 3.6.2. Python provides packages that have this validation method already implemented and

ready to use, but since the functionality is very simple, I decided to create it myself. This gives me more power to control what happens inside the validation.

The implemented validation splits the amount of files into given amount (k) of equal portions. Then it loops through the portions and trains the model, as described in the previous section 4.4.1, but leaves the files in the current portion out of the training files. These loops are called folds. In each fold, the left out files are then classified with the trained model. Then the fold accuracy is measured as the percentage of correctly classified texts. This can be measured because we already have the correct labels for those files. And in the end, the total accuracy of the model is the average accuracy of all folds.

The chapter 5.2 will show the validation results without preprocessing and with preprocessing. The results indicate that the k-NN classification is working as intended. They also show that the results can be further improved with preprocessing. Preprocessing is handled in the next section 4.5.

4.5 Implementing preprocessing

Even though the k-fold cross-validation at this point showed promising results, it was clear that they could be improved with text preprocessing. The basic preprocessing methods used in the training are lemmatisation, stop word removal, turning text to lowercase, removing unnecessary punctuation and, finally, tagging. The other methods are rather straight forward and easy-to-implement with existing libraries, but tagging needed more focus in order to work in the desired way. The other preprocessing methods are better described in the section 3.5. And the tagging implementation is described below.

4.5.1 Tagging implementation

The documents contain a lot of changing information, that is still relevant to the context, but machine learning models may have a hard time learning anything from the changing

information. For example, teaching the model that an IBAN is invoice information, is very relevant. But teaching the model that a specific IBAN is invoice information, is useless. The model should identify all future IBANs as invoice information. Therefore, tagging was introduced to replace these changing words with known tags. The first task was to identify which information should be tagged. Seven tags were created: `<url>`, `<bic>`, `<date>`, `<iban>`, `<email>`, `<number>` and `<amount>`.

The implementation works by splitting the text into an array, where one item is one word. Then it goes through the array and runs each word through an algorithm that tries to figure out if the word should be replaced with any of the predefined tags. The following section shows how each tag is handled.

Tagging numbers

Receipts and invoices contain a lot of changing numbers, so tagging them is quite important. Therefore, the number tagging was split into two different tags: `<number>` and `<amount>`. The `<amount>` tag is used to tag any numbers that reassemble some amount. Usually an amount of money. The `<number>` tag, on the other hand, tags all remaining numbers. The implementation tags all numbers that have a dot (.) or a comma (,) as the third last character as `<amount>`, for example "*100,00*" or "*0.89*".

Tagging banking information

Tagging international bank account numbers (IBAN) and business identifier codes (BIC) is useful to identify a document as an invoice. This is also important information for the user of this software, so the user knows where the invoice should be paid to. The implementation works by searching for a word that begins with two letters and is followed by a line of numbers. The found text is then validated with an IBAN validator, and correct IBANs are tagged with `<iban>`. BICs, however, are listed in a predefined list. All 8 character long words are compared with this list to correctly tag BICs. The current

implementation supports only Finnish BICs.

Tagging emails and web addresses

Emails are also tagged, since they are usually part of sender or receiver information. The application uses a Python package called `validate_email` to tag correct email addresses with `<email>`. Web addresses are tagged in a similar approach. The words are validated with a package `urlextract`, and tagged with `<url>`.

Tagging dates

Tagging dates is the most difficult task in the tagging process. Dates can be presented in various formats and in different languages. It would require a lot of effort to correctly tag all date formats – and the tagging mainly affects only to the due date detection. Therefore, a lot of effort wasn't put into the date tagging. The implemented method supports only the format "*D.M.Y*", for example "*12.3.2019*" or "*5.12.20*".

4.5.2 Using the implemented preprocessing methods

Once the preprocessing methods were implemented, they were introduced into the training and classification processes. The training data we have, is not preprocessed in any way – this is the common situation in machine learning tasks. Therefore, the preprocessing implementation must be run for each sentence we use in the training process, before training the model. And since the model is then trained with preprocessed data, the new data must also be preprocessed with the same implementation before running it through the classification.

After the preprocessing was utilized in the application, the model was once again tested with the same k-fold cross-validation method as earlier. Using the same data with the same process – other than preprocessing – revealed that the results did indeed improve with preprocessing. These results are represented in the section 5.2.

4.6 Classifying new documents

After the classification process proved to be successful, it was time to implement it into a more user friendly environment that can take in new documents and classify their content. This meant combining things from the previous tasks into one application.

The application would have a graphical user interface, that would very closely follow the GUI of the manual classification application from the chapter 4.3. Then it needs to take in training files and train the k-NN model as described in the section 4.4.1. And before the training is performed, the application needs to apply preprocessing to the text as described in the chapter 4.5. And the preprocessing needs to be performed on the new text files as well before the classification. Once all this works, the relevant information from the classified texts should be extracted. This extraction will be described in the section 4.7.

The next section 4.6.1 describes how the graphical user interface of the application works. It also contains a figure to help understand the layout. The functionalities of the application are explained deeper in the section after that – in section 4.6.2.

4.6.1 The graphical user interface

Classifying a document creates two files, one with the texts for each crop area from the document and one with the corresponding labels for these texts. These files don't really give any information for the user. Therefore, a graphical user interface was created for the classification application.

The application loops all documents in a given folder and all pages in each document. It shows the current page as an image on the left side of the application. The user then has the opportunity to either skip the page and move on to the next one or to process the page. Processing the page loops all crop areas in the document and classifies the areas with the trained k-NN model. After this, it will paint the areas on top of the document with a color

matching to the class.

The class names and colors are presented on the right side of the application. This allows the user to inspect which areas of the document were classified into which classes. The right side also displays the file name and the page number of the currently visible page. There is also a button to open that file with the operating system default PDF reader, since the image of the page might sometimes be a bit blurry.

The figure 4.4 shows an image of the GUI. In this figure, the shown page is already processed with the trained k-NN model, using three nearest neighbors. On the left side of the application is an image of the processed page with classified areas colored on top. The right side of the application shows the colors for each category. Above these colors we can see the name of the file, the page number and a button to open the document. This document can be found in the appendix A, figure A.1.

Inspecting the image shows us that the system gives promising results. We can see that the receiver and sender information at the top left section of the document, as well as in the invoice template at the bottom of the page, is classified into "*Receiver/Sender*" class (red). We can also see some product information around the middle section of the page (yellow), then total amount information below the products (green) and taxing information below that (purple). This is a very common structure for an invoice. There are also quite a few areas classified into the class 6, "*Invoice information*" (orange), but no areas classified into the class 5, "*Receipt information*" (blue). This is a good sign for an invoice.

4.6.2 Behind the GUI

When the application is executed, it will train the k-NN model with the provided folders of training texts and corresponding labels. Then it will go through the new documents in another provided folder. For each document, it will make an image of all the pages with a package called `pdf2image`. These images will be shown on the GUI when looping

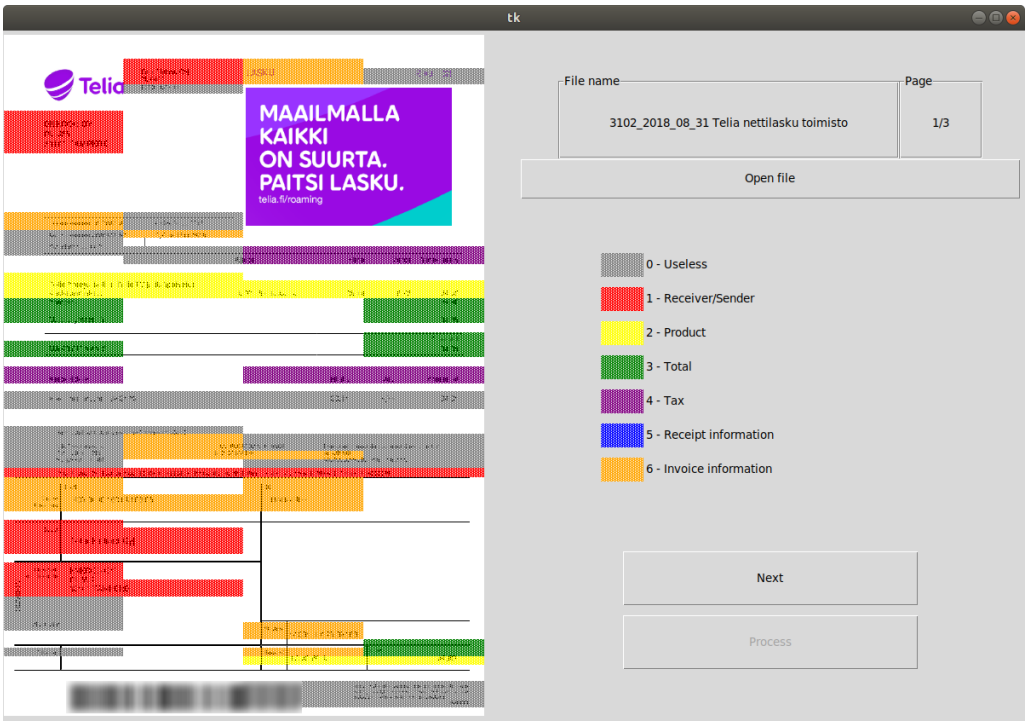

Figure 4.4: Graphical user interface of the automatic classification application.

through the pages. It will also detect the language or languages for each page with the help of `langdetect` package. These languages will be used for the stop word removal.

After the training is finished, the application will display the first page of the first document to the user. It will also update the file name and the page number correctly. If the user decides to process the visible page, the application will start going through the rows and columns of the page – the crop areas. For each crop area, the application will make a copy of the original PDF file, and crop that file into the crop area size. For the cropping, the application uses a package called `PyPDF2`. From that package, it uses a class `PdfFileReader`, which allows resizing the document size.

Once an area is cropped to the desired size, it is time to read and classify the text from the cropped PDF file. The text content is extracted with a library called `pdfminer`. Then the text is preprocessed as described in section 4.5. And after that, it is classified with the k-NN model that was trained when the application was started. This process is done for all crop areas until the whole page is processed.

When all the crop areas of the page are classified, the classified areas are drawn on top of the image. Then the user can move on to the next page to process it. Processing the next page starts the classification process from the beginning. After all pages of a document are processed, the classified texts can be saved to a new text file, where all texts from one class are grouped together. The next section 4.7 shows how these saved files will be used to extract the wanted information.

The following figure 4.5 shows a flowchart, that visualizes the basic process of the described application. In the beginning, the k-NN model is trained. After that, the PDF file pages are looped. If the user decides to process a page, all crop areas on the page are looped and possibly classified. If all pages of a document were processed, the results are saved into an output file. This is the basic process of the application. By changing the code settings, the user may, for example, change the class names, the class colors, the crop area size and the number of neighbors. The user can also decide to save the output files for each page separately. And even the graphical user interface can be hidden.

4.7 Extracting the needed information

After the classification process is completed for one whole file, the system will output a text file where the classified texts are grouped together. The file contains for example a line "*# 3 - Total*", and then below that line are listed all lines of text that belong to the class "*Total amount*". Now we still need to extract the actual total amount information from those lines of text.

The next 5 sections will go through each of the collected information types and show how they are extracted. After these sections, the section 4.7.6 will sum up how the extracted information could be further processed. The goal of the thesis, however, is reached after the data extraction is complete. Section 5.3 shows how well the system performs in extracting the desired data.

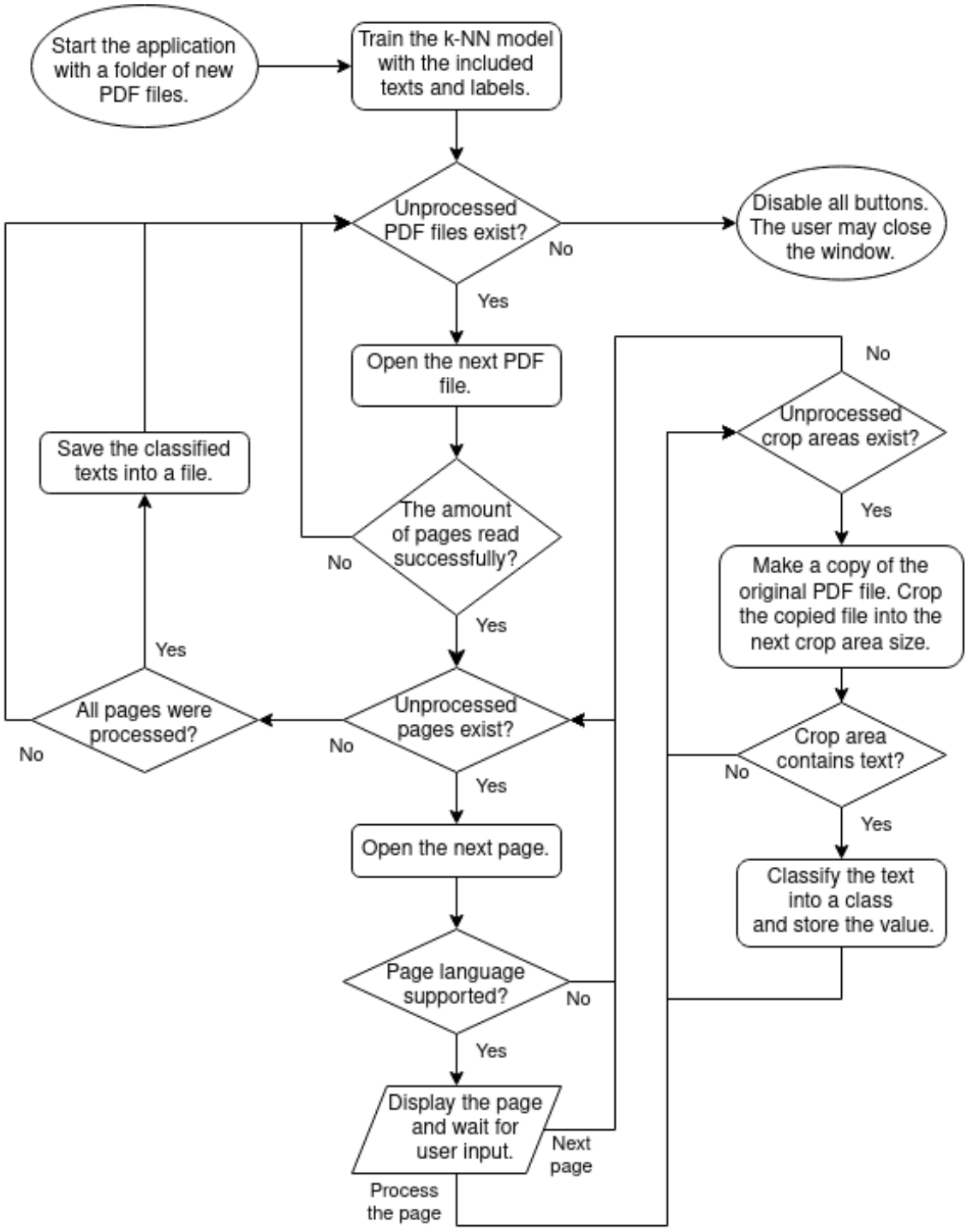

Figure 4.5: A flowchart for the classification application.

4.7.1 Sender information

The sender information is extracted from the class 1, "*Receiver / Sender information*". This happens by filtering the receiver information out of the text and then leaving only unique information in the returned text. The application must be provided with the receiver information before running the extraction script.

4.7.2 Product information

The second class, "*Product information*", is the most difficult class to process. Product information can contain almost any text, so it is hard to distinguish which information is relevant and which is not. And on the other hand, the outcome of this process is not very relevant for the final program. Therefore, the current process is a very simple implementation that only removes all duplicated lines of information. The actual names of the products are left for the user to identify.

4.7.3 Total amount and currency

Unlike the product information, the total amount information is very relevant in the aspect of this research. The implementation tries to find the total amount and the used currency of that amount from the class 3, "*Total amount information*". If the total amount isn't found in this class, the system will try to calculate it from the tax percentages as described in the next section 4.7.4.

The system supports only four currencies that are found in the provided test data – EUR, USD, GBP and SGD – but could be extended to support more. In order to find the used currency, the application counts all appearances of the used currencies or their symbols. Then the most used currency will be the currency for the total amount.

Finding the total amount works in a similar way. The application tries to find an amount in each word in the class. Then it makes sure the previous word, the current

word or the next word don't contain the percent sign (%), in order to make sure that the current amount isn't an incorrectly classified tax percentage. The application counts all appearances for each found amount. The final total amount is then the amount with the most appearances or the highest amount found.

4.7.4 Tax percentage and the amount verifications

The tax percentage information is collected from the class 4 – "*Tax information*". The application goes through each word in the class, and tries to find the percent sign (%) in the word. Then it will inspect that word and possible next or previous words. If an amount is found in them, it is marked as a tax percentage. Tax extraction also collects all amounts from the text, that aren't marked with the percent sign.

If the system didn't find any tax percentage with the process, but it did find a total amount in the previous process (4.7.3), it tries to calculate the tax percentage. This is done by looping through the found amounts from the class 4. The amount must be smaller than the total amount in order to be the tax percentage or the total amount without tax. If it's smaller than half of the total amount, assume it's the tax percentage. If it's higher, assume it's the total amount without vat. This assumption is made because tax percentage is very rarely above 50 %. Then the system calculates the percentage and the amount of tax based on these assumptions.

In the end, the tax percentage extraction will loop through all the found percentages and all the found amounts, and it will calculate the total amount, the amount of tax and the total amount without tax, for all amount and percentage combinations. Then it will try to find any of these calculated amounts among the found amounts. And after that, it will still go through the percentages that aren't yet identified as a tax percentage. It tries to calculate the correct amounts for those percentages based on the remaining untaxed amount. This complex process is done in order to verify the found amounts and percentages. And if some of them are missing, the system tries to calculate them and find the calculated

numbers in the text.

4.7.5 Document type and type specific information

One of the key objectives for this thesis is to identify the document type as a receipt or an invoice. For this purpose, the last two classes are used: "*Receipt information*" and "*Invoice information*". These classes contain also other information, specific to those classes. The system will first extract this information, and then the final decision is based on the found information and the length of the classes.

The receipt information extraction tries to find the charged payment card manufacturer and the last digits for the card. The implementation supports only "*Visa*" and "*Mastercard*" payment cards, but could easily be extended to other manufacturers also. First, the system tries to find those manufacturers in the text. Then it tries to find numbers next to those words or some hidden text marks, for example "****". If the manufacturer or the digits are found, these are included in the receipt information.

The system will then go through the invoice information to find information specific to invoices. These are the due date and the receiver IBAN. The due date extraction looks for predefined words, like "*due*". Then it tries to find a date next to that word. If such a date is found, it is marked as the due date. The IBAN extraction currently supports only Finnish IBANs. The system searches for words that begin with "*fi*". Then it tries to collect the following numbers and check that it is a correct IBAN. The found due date and IBAN are included in the invoice information.

If the system successfully found a due date or the receiver IBAN, it will assume that the document is an invoice. Or if it found the used card, it assumes the document is a receipt. If none of these information is found, the system will make the decision based on the amount of content in the classes "*Receipt information*" and "*Invoice information*".

4.7.6 Concluding the information extraction

Once all information from the provided text file is collected, the system will print the information for the user. It displays all information that it was able to extract. And it will also print a warning for the information that it was unable to extract. For the tax percentages, it prints all percentages and amounts related to that percentage. The output also tells whether the amounts are calculated or found in the text. The system always prefers the found amounts over the calculated amounts. Take a look at the appendix B to see the actual output from the processed example invoice in the appendix A. The chapter 5.3.1 will talk more about the extracted data from the example invoice.

Currently the following steps are left for the user to decide. The system does nothing else in addition to showing the extracted information. If the system is at some point further developed, the output could be directed to a file or into a database. Then the information would be available for other processing or for storage purposes. But the scope of the thesis ends here.

5 Results

This chapter groups together results from the different parts of the application. Each section in this chapter will refer to the chapter that the results belong to. The first section 5.1 shows how many PDF files were processed in total and how many of these files were processed successfully. It shows what type of errors the processing faced. The chapter will also show how much training data was successfully created from the PDF files. The second section 5.2 shows the results from the k-fold cross-validation method, which is used to measure the k-nearest neighbors model accuracy. It also shows how the preprocessing affected the results. The last section 5.3 shows how well the goal of the thesis is reached, i.e. how well the system classified documents into correct categories and how well all desired data is extracted.

5.1 Extracting training samples from the PDF files

As described earlier in section 4.3.1, reading the content of PDF files is sometimes problematic. Some files don't provide any content and some files provide only content that cannot be processed. This may happen, for example, because of a faulty or unsupported PDF file. This section describes how many of the available files were read and processed successfully, and why some of the files failed. The results in this section are closely related to the chapter 4.3.

This thesis utilized three different data sets of invoices and receipts. The first one consists of 60 receipts, and the second one of 52 invoices from Eneroc Ltd. The third one

Table 5.1: PDF file content reading success and failure rates.

	Data set 1	Data set 2	Data set 3	Total
Amount of files	60	52	224	336
Successfully read	30 (50 %)	34 (65 %)	165 (74 %)	229 (68 %)
Empty text content	28 (47 %)	17 (33 %)	29 (13 %)	74 (22 %)
Invalid text content	1 (2 %)	0 (0 %)	20 (9 %)	21 (6 %)
Unable to read file	1 (2 %)	1 (2 %)	1 (0 %)	3 (1 %)

is a mixture of 224 receipts and invoices from Eneroc Ltd. associates. This gives us the total amount of 336 documents. The first two data sets were mainly used for testing and validation, while the last data set was used for training and implementing the system.

The table 5.1 shows us the distribution of the amount of files and the amount of unsuccessful files among all data sets. The data sets are presented on the columns and the last column shows the total amount of all documents. We can see that the third data set is clearly the largest set. Please note that the three bottom rows for unsuccessful files may overlap with each other. Also, some files may have been skipped for various reasons, and therefore the percentages might not add up to 100 %.

The table shows that there are many documents where text content could not be extracted at all – visible in the row "*Empty text content*". These numbers contain mainly documents that have been scanned into the PDF file format, but haven't been processed with an OCR engine. The high numbers show us that good OCR solution is essential for the final application. The row "*Invalid text content*" contains files that were read correctly, but the extracted text content is unusable. The text contains for example unknown characters. The last row "*Unable to read file*" contains files that could not be read or opened at all. These files would require some special processing in the final application.

5.1.1 The amount of training data samples

The amount of files alone doesn't tell much about the amount of training data. Every successfully read file contained multiple cropped areas that were extracted into text and labeled. Each of these text and label pairs is one sample in the training data. The k-NN model was trained on the data set 3, which contains 165 successfully read files. These files produced the total amount of 15 783 training samples.

5.2 k-fold cross-validation results

This research used k-fold cross-validation to validate the trained k-NN model as described in the section 4.4.2. The model was first validated before any preprocessing methods had been implemented. And then it was validated again after the preprocessing methods were implemented. The validation shows that preprocessing indeed helps to improve the results. The k-fold cross-validation used all created data samples from the data set 3 – 15 783 samples.

The left side of the table 5.2 shows the validation results without preprocessing and the right side with preprocessing. The validation used 15 folds in both cases. The table shows the mean accuracy, the variance and the standard deviation of the 15 fold accuracies. Both cases used 5 nearest neighbors to determine the correct label. The accuracy is calculated by the amount of correctly labeled texts among all classified texts.

The table 5.2 shows us that there is an improvement after the preprocessing was introduced. The accuracy increased, while the variance and the standard deviation decreased. The results didn't improve much, but it shows us that preprocessing is useful. The results indicate that implementing better preprocessing methods might improve the results even more. Section 6.1.3 describes how the preprocessing could be further improved.

Table 5.2: k-fold cross-validation accuracy results with and without preprocessing. The tables show the mean (μ), the variance (σ^2) and the standard deviation (σ) values of 15 fold accuracies.

Amount of folds	15
Amount of neighbors	5
Preprocessing	No
μ	0.908558
σ^2	0.001021
σ	0.031956

Amount of folds	15
Amount of neighbors	5
Preprocessing	Yes
μ	0.917900
σ^2	0.000355
σ	0.018844

5.3 Data extraction results

This section describes how well the final solution of data extraction works. This means, for example, how many documents were labeled correctly as an invoice or a receipt, or how successful the system was on extracting the total amounts. The chapter 4.7 shows how the extraction implementation works.

The model was trained with the data set 3 and then tested on the data sets 1 and 2. The table 5.3 shows how well the system extracted the correct information from the data sets 1 and 2. The first data set contains 30 files and the second one 34 files that were successfully processed through the whole process of cropping, classifying crop areas and extracting information from the classified texts.

It is safe to assume that all processed documents contain at least the following information: "*Sender*", "*Products*", "*Total amount*", "*Tax percentage*" and "*Type (invoice/receipt)*". These information types are presented on the first 5 rows below the "*Amount of files*" row. The three bottom rows present information that is found only on certain amount of documents. Therefore, the percentages of these rows don't represent the actual success rates, since they are calculated from the total amount of files in that data set.

Inspecting the table, we can see that the process correctly classified 83 % of the documents. This indicates that the classification process is working, but it should be further

Table 5.3: Data extraction proportions

	Data set 1	Data set 2	Total
Amount of files	30	34	64
Correctly classified	21 (70 %)	32 (94 %)	53 (83 %)
Correct total amount	30 (100 %)	29 (85 %)	59 (92 %)
Correct tax percentage(s)	21 (70 %)	28 (82 %)	49 (77 %)
Partially correct tax percentage(s)	3 (10 %)	1 (3 %)	4 (6 %)
All products found	2 (7 %)	10 (29 %)	12 (19 %)
Products found partially	13 (43 %)	15 (44 %)	28 (44 %)
Sender correctly	17 (57 %)	31 (91 %)	48 (75 %)
Receiver IBAN found	4 (13 %)	27 (79 %)	31 (48 %)
Due date correctly	2 (7 %)	21 (62 %)	23 (36 %)
Used card found	6 (20 %)	1 (3 %)	7 (11 %)

improved. Currently the decision is based solely on the amount of receipt or invoice related information found. If neither information is found, the decision cannot be made. Perhaps the classification process should instead inspect the document as a whole to make the decision.

The total amount percentage is rather good (92 %). It is one of the most important pieces of information on the documents, so this is good. The results indicated that the "`<amount>`" tagging should be further improved. For example, the system failed to find the total amounts of "*USD 4000*" and "*3 199,00 EUR*". The first amount was not tagged as an amount since it has no decimal separator. And on the second amount, the system fails on the space character, and would tag this as "`<number> <amount> EUR`". The incorrect tagging is probably one of the reasons why the total amount extraction failed in some cases.

The tax percentage was found at least partially correctly in 83 % of the documents.

The partially found percentages happen for documents that include more than one tax percentage. The process found some correct percentages, but not all. It can also fail if the percentage isn't marked with a percent sign. For example, some documents have a table where the header says "*VAT (%)*", but the actual percentage is marked much lower on the table. The system fails to identify this as a tax percentage.

Products were correctly found only in 19 % of the documents and partially found in 44 %. The product extraction is something that should definitely be improved. In addition to the low percentages, the product information contains a lot of unnecessary information. However, this shows that the product extraction is working in the right direction, even though it was paid little attention to. To improve the percentages, a lot of training data would be required in order to correctly identify the products. Also, the location on the document should be used in the training phrase, since most product information is usually located in the middle section of invoices and receipts. Proper lemmatisation and stemming methods should also be introduced in all supported languages to improve the product percentages.

The sender information extraction has same problems as the product information extraction. The information can contain pretty much any text, and thus the class texts contain a lot of unnecessary information – even when the correct information is also present. The current email tagging implementation is good for sender email recognition, but this is not enough. To find the actual sender company or person name, some **Named-entity recognition** (NER) method should be used. This is a method of finding named entities in the text, for example a company name. Then it could be tagged as "`<company>`". This would greatly assist the machine learning algorithms to mark this correctly as sender information.

The last three rows on the table 5.3 show information that is present only in a part of the documents, but is still essential. The data set 1 contains mainly receipts and the data set 2 mainly invoices. The table shows us that the data set 1 contains more payment

cards than the second data set – common for receipts. Also, the second data set contains more IBANs and due dates than the first data set – common for invoices. This implies that these classes are working correctly, even though the total percentages are less than 50 %. Implementing better date tagger would greatly benefit the due date extraction.

5.3.1 Data extracted from the example invoice

The appendix A presents an example of a real invoice used in the making of this thesis. It also lists all information from the invoice that is relevant for the user. This is the information that should be extracted from the invoice. The appendix B, on the other hand, shows the output of the data extraction for the example invoice. This is the output that is printed to the user after the extraction process. User then has to decide what to do with the output.

The output tells that the document was correctly labeled as an invoice. The script also found the sender information rather well, but we can see that the sender information contains excessive lines as well. For example, the lines "*O T*" and "*R I*" should be removed from the output. The script has also marked the receiver post area incorrectly as sender information ("*33101 TAMPERE*"). This happened because the extraction script was provided with a different post area as the receiver post area. The product category successfully found all product related information. But this category also contains multiple excessive lines of text.

Taxing information was correctly marked for this document. We can also see that the currency is marked correctly ("*(EUR)*"). The script has also marked all amounts with the label "*(found)*", which means that all these amounts were found in the document. The other possible label is "*(calculated)*", which is shown for calculated amounts. The taxing information contains also the total amount that the tax percentage belongs to. We can see that the amount matches with the next category "*TOTAL*".

The additional information contains information typical for invoices. The due date

and the receiver IBAN were successfully found in the process. This information together was enough to determine that the document is an invoice. The script failed to find the reference number of the invoice.

6 Discussion and future

The need for the implemented system originated from the business world. My workplace, Eneroc Ltd., wanted to reduce the amount of working hours wasted on browsing through received invoices and receipts. They implemented an automated system for the task. The system, however, is a rather straightforward text-based system, so they were looking for a more sophisticated approach.

The goal for this thesis was to show that machine learning could be used to improve the results over the previous system. Even though the results look promising, the new system has many parts that could be even further improved. And the current system, as is, isn't yet ready to be used in production. This chapter will go through some steps that should be done in order to improve the results. And it will also tell how the final application could work and where it could be used in addition to the original purpose.

6.1 Improvements on the current system

The current system has a few pieces that should be improved or implemented. Most of them aren't yet finished because they have little impact on the outcome of the application, or they fall out of the scope of this thesis. This section goes through those steps that should be completed in the next stage of this research.

6.1.1 From k-NN to machine learning

One of the key improvements that should be done on the product, is changing the k-nearest neighbors algorithm into an actual machine learning algorithm. k-NN is a good method to prove that the use of a machine learning approach is profitable. But k-NN itself isn't a machine learning algorithm, as we learned in the section 3.3.2.

The next steps in turning this system into a machine learning system would be testing different machine learning approaches to see which model performs best on our data. The current approach uses only accuracy as the performance measurement. Testing the machine learning models should use other, more sophisticated testing methods in addition to accuracy, in order to find out which model performs best.

6.1.2 Error handling

Error handling was included in multiple places when creating the application. But it could still be improved. PDF files are rather complex files to read and handle. Even libraries that are intended for PDF file reading some times fail to read the content. The current error handling mainly focuses on handling these situations so that the program doesn't crash.

The error handling should be improved so that the failing files are reprocessed with some other method. The other method might succeed where the first one failed. For example, if the file content cannot be read with one library, another library should be tested instead. Or if the content contains multiple unknown characters, perhaps another encoding method should be tested.

Unfortunately, there are also files that cannot be read properly. Some files are for example encrypted so that they cannot be read. The system should however be able to handle these files as well. As a last resort, the system could convert these documents into images. Then the images would be processed with an OCR system to extract the text content.

6.1.3 Improving the training phase

Text location in the document

Training the machine learning model has also sections that could be further improved – starting from the creation of the training data. The current approach reads the texts from each cropped area and classifies the text, but the text location on the document is lost in the process. This is a rather serious flaw on the data creation process.

Imagine yourself reading through an invoice, searching for the total amount to pay. You would most likely focus your eyes on the bottom right section of the document – this is a very common location for the total amount. The invoice information, like the receiver IBAN, would most likely be located on the bottom section as well. The invoice items would be listed in the middle section of the invoice. And the sender and receiver information would most likely be located at the top section of the document. Therefore, the location of the information is very essential on finding the desired information. And in addition to the text location on the document, the page number and number of pages is also essential.

Preprocessing improvements

The current system is designed specifically for the training data it uses. This means, the system supports only languages and currencies found in the training data. For example, lemmatisation is supported only for English language. Support for other languages should be added in the preprocessing. And in addition to lemmatisation, stemming should also be introduced into the system. Stemming is the process of turning all conjugated words to their base form.

The current tagging approach should also be improved. Especially the date tagging. The current implementation supports only "*D.M.Y*" format – e.g. "*16.3.2019*" or "*16.03.19*". Instead, some date formatting library should be used here to correctly tag dates with the tag "`<date>`". Also the Business Identifier Code (BIC) tagging should be

improved. Currently it supports only Finnish BICs listed in a predefined array.

6.1.4 Data extraction improvements

The current data extraction implementation works by extracting the wanted data from a group of classified texts. The extraction works well for simple information – like the total amount and the tax percentage – but struggles with more complex information – like the product list and the sender information. These areas are something that should be improved in the future. The section 5.3 goes through the different data groups and shows how well the data extraction worked. This section also describes many improvements that would help on the extraction.

Another improvement on the data extraction is the final output of the extracted data. The current output is just a plain text output, printed on the console. Take a look at the appendix B to see an actual output of the extraction script. Currently the user has to decide what to do with the output. In the future, the output should instead be exported in a machine readable format. The output could then be printed in a user friendly interface or be directed into a database.

6.2 The final application

6.2.1 Deploying the system

The current system is hosted on a private Gitlab repository, and requires installing multiple Python libraries in order to work – this isn't very convenient. Instead, it should require minimal effort from the user to start using the system. As the system is implemented specifically for the use of Eneroc Ltd., the simplest deployment method would be to host it on a company server.

Hosting the system on a server means that users could just upload files for processing, and then the server would return the results. The user wouldn't have to install anything or

know how the system works. The user interface could be a website, for example.

Another great thing about server hosted system is that the classification model is also stored on the server. User doesn't have to train anything nor download any model on his device. This guarantees similar results on every device since the used model is always the same. Hosting the system on a server also makes it easier to do modification on the system or the model.

6.2.2 Using the system

Once the system is up and running, users would probably use it via some website. On the website, users should be allowed to upload a file – or a group of files – of any supported file format (PDF, image or XML) to the server. The server then processes the file and returns the results in a similar format regardless of the original file format. The web page would then show these results for the user. The web page should also allow exporting this information or storing it in a database.

In case the system fails to extract some information or classifies something incorrectly, the user should have the opportunity to correct these mistakes. These errors could be uploaded to the server as well, and then they could be used to train the model more.

6.2.3 Who uses the system?

The system is designed to be used at Eneroc Ltd., where it would be used by someone who handles the incoming receipts and invoices. But even though this system is designed for such a specific use, it could be extended for other users as well.

After the system is working, access to it could be easily allowed to other companies as well. Since the system runs on a server, the other companies could use via the same website. In this case, security matter should be taken into account.

Individual users could also benefit from a system like this. It could be used to track money usage or simply to store receipt and invoice information. If the system proves to

work correctly, it could be extended for individual users as well.

7 Conclusion

The goal of this research was to find out if a machine learning based approach could be used to replace an old text based approach in a document classification task. The task is to classify documents into invoices and receipts. And in addition, some key information from the documents should be collected in the process.

The implemented k-NN based application correctly classified most of the new documents in a validation data set. It also collected the key information rather well. This confirms that machine learning based application would perform well on the given task.

The following section goes through the whole process that was carried out in the research. It summarizes the main sections of the research and shows where these sections are better described in this document.

7.1 Summary

The research began with roughly 300 invoice and receipt documents, which were better described in the section 2.2. The first task was to create suitable training data, so that a machine learning model could learn from the data. An application was created to manually classify document areas into predefined categories. The categories were presented in the section 3.4 and the application in the section 4.3. These categories represent different key information that should be collected from the document – for example, one of the categories is for the total amount related information. Training data was successfully created from 165 documents, which resulted in 15 783 training samples.

Instead of jumping directly to heavy machine learning models, a simpler k-nearest neighbors method was implemented to see if the use of a machine learning model would be beneficial. This implementation was described in the section 4.4.1. Some preprocessing was also introduced in the training process to improve the results, as described in the section 4.5. The trained k-NN model was validated with k-fold cross-validation – described in the section 4.4.2. With preprocessing implemented, the average accuracy of 15 folds was 0.9179. This shows that the k-NN model is working correctly, and machine learning approach ought to be implemented in the future. The k-NN results were presented in the section 5.2.

In order to use the trained k-NN model to classify completely new documents, another application was created, as described in the section 4.6. This application classifies document areas into the predefined categories with the k-NN model. Then it groups all texts from one class together to make further processing easier. These groups of text lines are then run through a script that extracts needed information from the classes. This extraction script was described in the section 4.7. The extraction works rather good on simpler information like the document type, the total amount and the tax percentage, but struggles a bit with the more complex information like the product list. The extraction results can be found in the section 5.3. Additional preprocessing, machine learning model and greater amount of training data should improve the results. Future improvements were represented deeper in the section 6.1.

References

[1] S. Bassi, *Python for Bioinformatics*, English. London: CRC Press LLC, 2017, pp. 237–238, ID: 4941279, ISBN: 9781351976961.

[2] A. S. Incorporated and J. King, *Document management — Portable document format — Part 1: PDF 1.7*. Adobe Systems Incorporated, Jul. 2008, pp. vi–viii. [Online]. Available: https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf.

[3] R. Brunelli, *Template matching techniques in computer vision theory and practice*. Wiley, 2009, p. 3, ISBN: 1-282-12346-7.

[4] A. Chaudhuri, S. K Ghosh, K. Mandaviya, and P. Badelia, *Optical character recognition systems for different languages with soft computing*, ser. Studies in Fuzziness and Soft Computing. Springer, 2017, vol. 352, pp. 1–3, ISBN: 3319502514. DOI: 10.1007/978-3-319-50252-6.

[5] Y. Zhang, S. Lu, X. Zhou, M. Yang, L. Wu, B. Liu, P. Phillips, and S. Wang, "Comparison of machine learning methods for stationary wavelet entropy-based multiple sclerosis detection: Decision tree, k-nearest neighbors, and support vector machine", Simulation, vol. 92, no. 9, pp. 861–871, 2016, ISSN: 0037-5497. DOI: 10.1177/0037549716666962.

[6] S. Alam and N. Yao, "The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis", Computational and mathemati-

cal organization theory, vol. 25, no. 3, pp. 319–335, 2018, ISSN: 1572-9346. DOI: 10.1007/s10588-018-9266-8.

[7] G. Dougherty, *Pattern Recognition and Classification: An Introduction*. Springer New York, 2013, pp. 162–163, ISBN: 9781461453239. DOI: 10.1007/978-1-4614-5323-9.

[8] "Embedded abbyy ocr drives new toshiba tec multi-function printers", Entertainment Close-up, Jun. 2016.

[9] G. Hackeling, *Mastering machine learning with scikit-learn: apply effective learning algorithms to real-world problems usung scikit-learn*. Packt Publishing, 2014, pp. 81–83, ISBN: 1-78398-837-1.

Appendix A Example invoice

The following page contains an image (A.1) of an actual invoice used in this thesis. The invoice is provided by Eneroc Ltd. Some of the fields are blurred for privacy reasons, but the implementation uses an uncensored version. The invoice is in Finnish language, which is typical for the invoices used in the research. Some key information from the invoice is listed here:

Type	Invoice
Sender	Telia Finland Oyj
	PL 0400 15101 LAHTI
Total amount	34,89 €
Tax percentage	24 %
Product	Telia Yhteys kotiin, Telia TV ja lisäpalvelut
	Kuukausimaksut 16.07.18 - 15.08.18
Sender IBAN	FI33 8000 1601 0166 95
Due date (DD.MM.YYYY)	17.09.2018
Reference number	*censored*

Telia Finland Oyj
PL 0400
15101 LAHTI

LASKU Sivu 1 (2)

ENEROC OY
PL 555
33101 TAMPERE

Laskun numero [redacted] Päiväys 31.08.2018
Asiakasnumero [redacted] Eräpäivä 17.09.2018
Yksilöintitieto [redacted]

	Ajalta	Hinta	Verot	Yhteensä €
Telia Yhteys kotiin, Telia TV ja lisäpalvelut				
Kuukausimaksut	16.07.18 - 15.08.18	28,14	6,75	34,89
Yhteensä				34,89
Maksut yhteensä				34,89
				Yhteensä €
MAKSETTAVA €				**34,89**

Veroerittely	**Hinta**	**Alv**	**Yhteensä**
Verollinen myynti, alv 24 %	28,14	6,75	34,89

Huomautukset laskusta on tehtävä ennen eräpäivää.

telia.fi/yhteydenotto
Puh. 020 693 693
Ma–pe klo 8–16.30

IBAN: FI3380001601016695
BIC: DABAFIHH

Perintäkulut saatavien perinnästä annetun lain mukaisesti.
Viivästyskorko korkolain mukaan.

Telia Finland Oyj, Teollisuuskatu 15, 00510 HELSINKI. Kotipaikka: Helsinki. Puh. 020 401. Y-tunnus: 1475607-9, Alv rek. FI14756079

TILISIIRTO

Saajan tilinumero	IBAN FI33 8000 1601 0166 95	BIC DABAFIHH	
Saaja	**Telia Finland Oyj**		
Maksajan nimi ja osoite	ENEROC OY PL 555 33101 TAMPERE		
Allekirjoitus		Viitenro	[redacted]
Tililtä nro		Eräpäivä 17.09.2018	Euro **34,89**

[redacted]

Maksu välitetään saajalle vain Suomessa Kotimaan maksujenvälityksen yleisten ehtojen mukaisesti ja vain maksajan ilmoittaman tilinumeron perusteella.
PANKKI

Figure A.1: An example image of an actual invoice used in the research.

Appendix B Data extracted from the example invoice

This appendix shows the extracted data from the example invoice in appendix A. The following code is the output of the data extraction script. Some parts of the output are slightly modified to make sure all data is clear and visible on the page.

```
FILE:
    ../Automatic_classification/output_text/3102_2018_08_31
Telia nettilasku toimisto.txt

RECEIPT OR INVOICE:
    Invoice

SENDER:
    Telia Finland Oyj
    0400 15101 LAHTI
    PL 555
    33101 TAMPERE
    Teollisuuskatu 15, 00510 HELSINKI. Kotipaikka: Helsinki.
Puh. 020 401. Y-tunnus: 1475607-9, Alv rek. FI14756079
    Saaja
```

Maksajan nimi ja osoite

O T

R I

Pääkäyttäjätunnus eneroc-1/ENEROC OY

PRODUCTS:

Telia Yhteys kotiin, TV ja lisäpalvelut

Kuukausimaksut

16.07.18 - 15.08.18

28,14

6,75 34,89

Yhteensä

17.09.2018

31.08.2018

ASENNUSOSOITE KOUSANKATU 1, 20610 TURKU

KUUKAUSIMAKSUT

kotiin XL Kaapeli 80-200M

Etuhinta EUR/kk, Hinnastohinta 39,90 EUR/kk 16.07.

15.08. 1

32,18

Määräaikainen sopimus voimassa 26.10.2019 asti*

TAX:

Percentage (%)	Amount of tax (EUR)
24.0	6.75 (found)
Total without tax (EUR)	Total (EUR)
28.14 (found)	34.89 (found)

```
TOTAL:
    34.89 EUR (found)

ADDITIONAL INFORMATION:
    Due date: 17.09.2018
    Receiver IBAN: FI33 8000 1601 0166 95
```